Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Triennale in Ingegneria informatica

Studio della betweenness centrality e valutazione sperimentale dell'algoritmo SILVAN su reti complesse

Candidato:
Alberto Penzo
Matricola 2020039

Relatore:
Prof. Leonardo Pellegrina

Correlatore:
Prof. Fabio Vandin

Anno Accademico 2022-2023

Abstract

La betweenness centrality (BC) è una metrica che misura la centralitá dei nodi in una rete, ovvero quanto sono importanti o centrali per i cammini minimi di un grafo. Nello studio e analisi di reti, la betweenness centrality é una metrica particolarmente importante perché permette di individuare i nodi chiave, ad esempio nella diffusione delle informazioni in un grafo. In questa tesi si fornisce una descrizione dell'algoritmo per il calcolo esatto della BC e si introducono degli algoritmi rigorosi per il calcolo approssimato della BC, i quali sono piú efficienti dal punto di vista computazionale e quindi preferibili per lo studio di reti complesse e di grandi dimensioni. Con particolare attenzione la tesi prende in considerazione SILVAN, un algoritmo in grado di stimare la BC utilizzando la tecnica del progressive sampling e concetti avanzati di statistical learning. Vengono inoltre analizzati alcune reti di grandi dimesioni mostrando come SILVAN é uno stimatore efficiente e rigoroso della BC.

Indice

Introduzione

Negli ultimi anni si è visto un forte aumento dell'interesse verso lo studio di reti di dimensioni sempre maggiori. Queste reti permettono di astrarre il funzionamento e la struttura di molti sistemi complessi, quali le reti sociali (reti di amicizie, reti di citazioni, social networks), i sistemi biologici (reti di interazione tra proteine e molecole), i sistemi di comunicazione (internet, connessioni peer-to-peer) e di trasporto (reti stradali, reti ferroviarie). Nello studio e analisi delle reti, un problema di fondamentale importanza è individuare nodi centrali o importanti. Per questo motivo sono state introdotte diverse misure di centralità. La centralità di un nodo nella rete, infatti, è particolarmente utile in quanto permette di studiare diversi aspetti delle reti, quali la resilienza di una infrastruttura a degli attacchi informatici o la capacità di un nodo nel controllare le informazioni che passano lungo i cammini della rete. In entrambi i casi sopra citati, identificare nodi centrali rappresenta un problema cruciale. Da qui nasce l'interesse e l'esigenza di sviluppare algoritmi in grado di calcolare con precisione ed efficienza diverse metriche di centralità.

Alcune di queste metriche forniscono informazioni sulle proprietà locali del grafo, altre invece forniscono informazioni sulla struttura complessiva del grafo. Per esempio, una delle piú semplici metriche di centralità è la degree centrality, che rappresenta per ogni vertice della rete il numero di vertici ad esso, connesso ovvero il grado di un vertice.

In questa tesi siamo interessati alla *betweenness centrality*, una misura di centralità piú raffinata basata sui cammini minimi. Questa misura attribuisce maggiore importanza ad ogni nodo nel grafo sulla base di quanti cammini minimi passano attraverso di esso. Dato che il calcolo della betweenness centrality diventa infattibile per grafi di dimensioni molto grandi a causa delle eccesive risorse richieste, diversi lavori di ricerca recenti hanno sviluppato degli algoritmi di approssimazione, i quali ottengono una stima di questa metrica. Questi algoritmi si basano sul campionamento casuale (*random sampling*) di cammini minimi e offrono una garanzia probabilistica sulla qualità dell'approssimazione. La Sezione 1.2.1 introduce il concetto di betweenness centrality in maggiore dettaglio, mentre

le Sezioni 1.4.4 e 2.2 descrivono diversi algoritmi sia per il calcolo esatto sia per quello approssimato. La Sezione 3.5 presenta una valutazione sperimentale delle performance di SILVAN su delle reti provenienti da applicazioni reali.

Capitolo 1

Definizioni preliminari

Questo capitolo introduce le nozioni fondamentali della teoria dei grafi, necessarie per introduttre gli argomenti che verranno discussi in seguito.

1.1 Grafi

In matematica un *grafo* è un tipo di struttura dati che rappresenta un insieme di oggetti chiamati *nodi* o *vertici* e le relazioni tra di essi chiamati archi o spigoli. Formalmente, un grafo G è definito come una coppia (V, E), dove V è l'insieme dei nodi e E è l'insieme degli archi, ciascuno dei quali collega due nodi. Gli archi possono essere non direzionati (grafi non diretti), ovvero percorribili in entrambi i versi, o direzionati, ovvero percorribili in un solo verso (grafi diretti). Inoltre i grafi possono essere pesati.
I grafi pesati sono una tipologia di grafo a cui ogni arco è associato un peso o valore numerico. I pesi possono rappresentare diverse informazioni come la distanza tra due punti o il costo di attraversamento.
In un grafo, un *cammino* è una qualsiasi sequenza di vertici tale che ogni coppia consecutiva di vertici nella sequenza sia connessa da un arco nella rete. I cammini possono essere definiti sia per grafi diretti che non diretti. In un grafo diretto ogni arco attraversato da un percorso deve essere attraversato nella direzione corretta. In un grafo non diretto, gli archi possono essere attraversati in entrambe le direzioni. Un *cammino minimo*, detto anche percorso geodesico, tra due nodi di un grafo è il cammino che contiene il minor numero di archi. Se il grafo è pesato allora il cammino minimo è definito come il percorso che minimizza la somma dei pesi degli archi. La lunghezza del cammino è chiamata distanza geodesica. Quando si parla di distanza tra due nodi ci si riferisce alla distanza geodesica tra di essi. Possono esistere piú cammini minimi tra due nodi del grafo, tuttavia

la loro distanza geodesica è ben definita in quanto tutti i cammini minimi tra gli stessi due nodi hanno per definizione la stessa lunghezza. Esistono vari modi di rappresentazione dei grafi. La *matrice di adiacenza* è una delle rappresentazioni più comuni. In questa rappresentazione, un grafo con n nodi viene rappresentato come una matrice quadrata A di dimensione $n \times n$, in cui l'elemento $A[i, j]$ indica se esiste un arco diretto dall'i-esimo nodo al j-esimo nodo. Se il grafo è non diretto, la matrice di adiacenza sarà simmetrica rispetto alla diagonale principale. In questo caso, la matrice di adiacenza contiene solo informazioni sulla presenza degli archi, non sul peso degli stessi. Se il grafo è pesato, ogni elemento della matrice di adiacenza conterrà il peso dell'arco corrispondente.

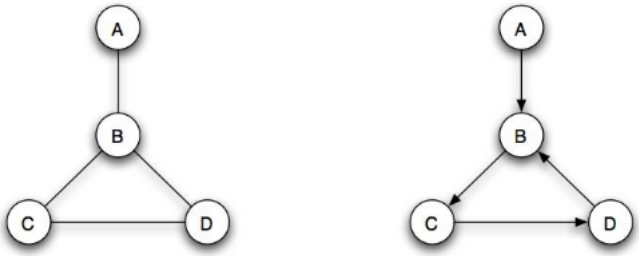


Figura 1.1: Grafo non diretto (sx) vs grafo diretto (dx)

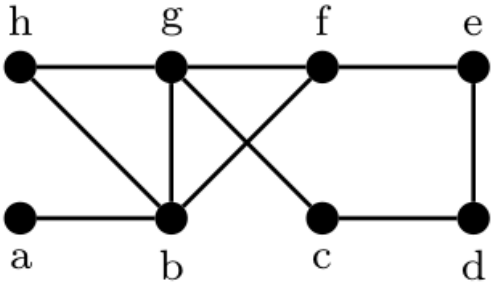


Figura 1.2: Esempio di grafo. Un cammino potrebbe essere il percorso $c - d - e - f$ che collega il nodo c al nodo f. Il cammino minimo tra c ed f è $c - g - f$

La matrice di adiacenza è una rappresentazione semplice e intuitiva dei grafi, ma può diventare poco pratica per grafi molto grandi o sparsi (un grafo è sparso se il numero di archi è dello stesso ordine di grandezza del numero di vertici ovvero $m \in \mathcal{O}(n)$), in quanto la maggior parte degli elementi della matrice sarà nulla. Inoltre, la matrice di adiacenza richiede spazio proporzionale a n^2, il che può rappresentare un problema per grafi di grandi dimensioni.

Per ovviare a tali limitazioni una soluzione alternativa è rappresentare il grafo utilizzando una *lista di adiacenza*. In questa rappresentazione, per ogni nodo del grafo viene memorizzata una lista dei suoi nodi adiacenti. Questo permette di ridurre lo spazio di archiviazione necessario per il grafo; questo però incrementa il costo necessario per verificare l'esistenza di un arco tra due vertici s e t, il quale è costante utilizzando la matrice di adiacenza, ma lineare nel grado di s tramite le liste di adiacenza.

1.2 Misure di centralità

Molte ricerche si sono concentrare sul concetto di centralità, ovvero su quali siano i vertici piú importanti o centrali di un grafo. Questa sezione descrve alcune delle misure di centralità piú comunemente utilizzate.

1.2.1 Degree centrality

La *Degree centrality* di un nodo anche detta *grado* di un nodo è il numero di archi connessi ad esso. Nelle reti dirette, i vertici hanno sia un grado entrante (*in-degree*) che uno uscente (*out-degree*). Per un grafo indiretto $G = (V, E)$ con $n = |V|$ nodi e $m = |E|$ archi, il grado k_i di ciascun nodo i può essere espresso dalla seguente formula:

$$k_i = \sum_{j=1}^{n} e_{i,j}. \tag{1.1}$$

dove $e_{i,j}$ è uguale a 1 se esiste. Per i grafi diretti la degree centrality k_i di un nodo i viene definita come la somma del numero di archi entranti e uscenti:

$$k_{tot} = k_i^{in} + k_i^{out}. \tag{1.2}$$

dove k_i^{in} è il numero di archi entranti nel nodo i e k_i^{out} è il numero di archi uscenti dal nodo i.

1.2.2 Closeness centrality

La *Closeness centrality* di un nodo è una misura di centralità in una rete che indica quanto sia vicino a tutti gli altri nodi del grafo. In altre parole, un nodo con una closeness centrality più alta è piú vicino a tutti gli altri nodi nel network rispetto a un nodo con una closeness centrality più bassa. Definiamo d_{ij} la lunghezza di un cammino minimo da

i a j. Allora la distanza geodesica media da i a j, calcolata su tutti i vertici j nel grafo è:

$$l_i = \frac{1}{n-1} \sum_j d_{ij}. \tag{1.3}$$

Questa quantità assume valori piccoli per vertici che sono separati dagli altri da una distanza gedesica breve in media. Siccome l_i ritorna valori piccoli per i nodi piú centrali la closeness centrality C_i di i è definita come l'inverso di l_i:

$$C_i = \frac{1}{l_i} = \frac{n-1}{\sum_j d_{ij}}. \tag{1.4}$$

1.2.3 Betweenness centrality

Consideriamo un grafo $G = (V, E)$ con $n = |V|$ (numero di vertici), $m = |E|$ (numero di archi).

Siano s e t due vertici del grafo G, definiamo con σ_{st} l'insieme dei cammini minimi tra i due vertici e $\sigma_{st}(v)$ il sottoinsieme di σ_{st} contenente tutti i cammini minimi che attraversano il vertice v.

La betweenness centrality (BC) $b(v)$ per il nodo v è definita come il numero di volte in cui v si trova sul percorso più breve tra due altri nodi s e t del grafo, considerando tutte le possibili coppie s e t con $s \neq t$, ovvero In formula abbiamo:

$$b(v) := \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{1.5}$$

Una versione della BC normalizzata si può calcolare dividendo l'Equazione 1.5 per il numero totale di coppie di vertici, ovvero $n(n-1)$:

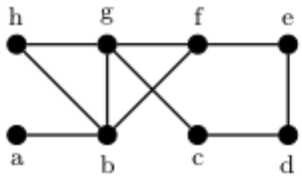
$$b(v) = \frac{1}{n(n-1)} \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{1.6}$$

In questo modo la BC risultante sarà compresa nell'intervallo [0,1] e rappresenterà la frazione (piuttosto che il numero) di cammini minimi che attraversano un determinato vertice. La versione normalizzata di $b(v)$ è preferibile per confrontare la centralità dei nodi in diversi grafi o in grafi di diverse dimensioni.

Un esempio di valori di BC per tutti i vertici di un grafo non diretto con 8 vertici e 10 archi è dato dalla figura 1.3

1.3 Osservazioni

Di seguito riportiamo alcune proprietà interessanti della betweenness centrality.



(a) Example graph
(b) Betweenness values

Vertex v	a	b	c	d	e	f	g	h
b(v)	0	0.250	0.125	0.036	0.054	0.080	0.268	0

Figura 1.3: Esempio di valori di betweenness centrality per un grafo non diretto (da [9])

- La BC può essere calcolata sia per grafi diretti che per grafi non diretti; nel caso di grafi diretti, i cammini minimi tra due vertici potrebbero essere diversi a seconda che questi partano da a o da b, ma ciò non influisce sulle Equazioni 1.5 e 1.6 in quanto includono esplicitamente i cammini percorsi in entrambi i versi per ogni coppia di vertici.

- La BC è una metrica di centralità che considera l'importanza di un nodo a livello dell'intero grafo a differenza di altre metriche (e.g. *Degree centrality*), le quali considerano la connettività di un nodo solamente a livello locale. Ad esempio, osservando la figura 1.4, si nota che il nodo v funge da "ponte" tra la regione C1 e la regione C2. Ogni cammino che parte da un nodo che si trova in C1 e termina in nodo in C2 deve per forza passare per v, quindi è ragionevole pensare che v abbia un alto valore di BC nonostante la sua connettività sia molto bassa (e.g. *Degree centrality* è solo 2).

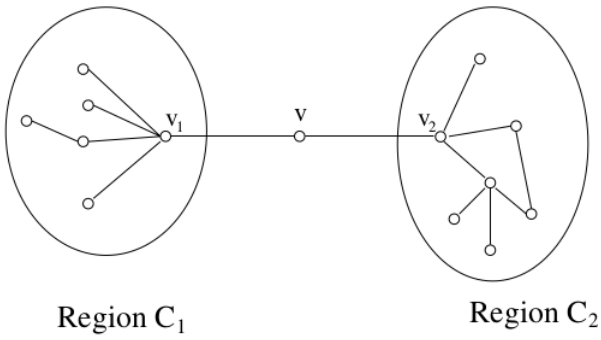


Figura 1.4: Il nodo v funge da ponte tra la regione C1 e la regione C2(da [1])

- Le equazioni 1.5 e 1.6 si applicano anche per grafi con piú di una componente connessa. Infatti se s e t si trovano su due componenti connesse diverse, non esistono cammini minimi tra di essi e il loro contributo varrà zero. Invece, altre misure di centralità, come ad esempio la closeness centrality, non possono essere applicate a grafi non connessi.

- I valori che può assumere la BC sono influenzati dalla topologia del grafo e possono assumere valori molto diversi tra di loro. Il massimo valore di BC per un nodo si verifica quando quel nodo giace sui cammini minimi di tutte le altre coppie di nodi del grafo, consideriamo, ad esempio, un grafo a stella. Il nodo interno di un grafo a stella (un grafo a stella è un grafo bipartito completo, ovvero è possibile partizionare il grafo in due sottoinsiemi U e V dove ogni nodo in U è connesso ad ogni nodo in V e non vi è alcun arco tra due vertici in U e due vertici in V) rispetta questa proprietà in quanto giace su tutti gli n^2 cammini minimi tra le coppie di nodi eccetto gli $n-1$ cammini minimi dei nodi esterni con loro stessi. Per cui la BC del nodo interno vale $n^2 - n + 1$ (massimo valore BC)[7]. Il minimo valore di BC in un grafo può essere zero se non esiste alcun cammino minimo tra nessuna coppia di nodi che passa attraverso quel nodo. Un esempio sono i nodi esterni del grafo a stella.

Figura 1.5 mostra l'esempio di un grafo a stella con $n = 9$ nodi. Si può notare che il nodo evidenziato in rosso, attraversato da tutti i cammini minimi che collegano due nodi neri distinti, è il nodo di massimo valore di BC, mentre i nodi neri hanno BC pari a 0, dato che non vi è alcun cammino minimo che li attraversa.

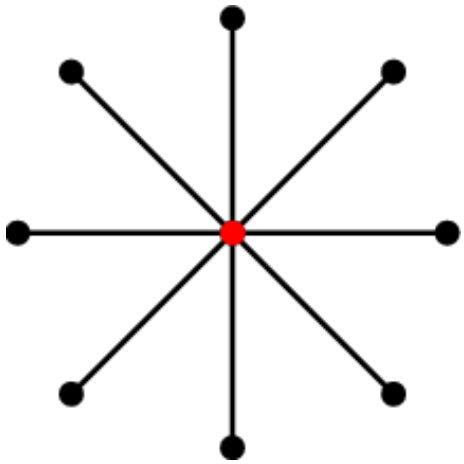


Figura 1.5: Grafo a stella con 9 nodi. Il nodo evidenziato in rosso ha il massimo valore di BC mentre i nodi esterni hanno BC pari a 0

1.4 Calcolo esatto della betweenness centrality

Come già visto il calcolo della BC richiede come step preliminare la ricerca dei cammini minimi tra tutte le coppie di nodi di un grafo. Per risolvere questo primo passaggio introduciamo due problemi computazionali: SSSP (Single Source Shortest Path) e APSP (All-Pairs Shortest Path).

1.4.1 Single Source Shortest Path

Nel problema SSSP siamo interessati a calcolare le distanze geodesiche tra un nodo specifico e tutti gli altri nodi del grafo.
L'algoritmo di Dijkstra è in grado di risolvere questo problema per grafi pesati con pesi non negativi in tempo $\mathcal{O}((n+m)\log n)$.[4] Di seguito riportiamo una breve descrizione dell'algoritmo e lo pseudocodice.

L'algoritmo di Dijkstra per il calcolo dei cammini minimi da un singolo nodo sorgente s a tutti gli altri nodi del grafo può essere implementato utilizzando una priority queue. L'algoritmo utilizza, per ciascun vertice v del grafo, un'etichetta che rappresenta la stima corrente della distanza da s a v. Inizialmente, l'etichetta di ogni nodo da s viene impostata a infinito, tranne che per il nodo s che ha etichetta 0. I nodi vengono poi inseriti nella priority queue, ordinati in base alla loro etichetta. In ogni passo, il nodo v con l'etichetta più bassa viene estratto dalla priority queue. Per ogni vertice w vicino di v, se la somma della sua etichetta e il peso dell'arco tra v e w è minore dell'etichetta corrente di w, allora l'etichetta di w viene aggiornata e w viene inserito nella priority queue. Il processo continua fino a quando tutti i nodi sono stati estratti dall priority queue. Quando un nodo viene estratto dalla priority queue, la sua etichetta corrente è la lunghezza del cammino minimo dal nodo sorgente al nodo stesso. Inoltre, per ogni nodo visitato, la sua etichetta corrente è la stima migliore della lunghezza del cammino minimo dal nodo sorgente al nodo stesso. L'algoritmo ha una complessità temporale $\mathcal{O}(m + n\log n)$. Se gli archi non sono pesati, l'algoritmo si riduce ad una BFS[5] con complessità temporale $\mathcal{O}(m+n)$. Inoltre, l'algoritmo permette di attraversare i cammini minimi da s a tutti gli altri vertici mediante dei campi *pred*; infatti, $pred(v)$ è un puntatore al nodo precedente a v nel cammino minimo da s a v. Il cammino minimo può essere ricostruito chiamando ricorsivamente la funzione finche non restituisce il nodo s.

Algorithm 1 Algoritmo di Dijkstra per SSSP

Input: Grafo $G = (V, E)$, pesi archi $\omega : E \longrightarrow \mathbb{R}$, nodo sorgente $s \in V$

Output: Distanze geodesiche $d(s, v)$ per ogni $v \in V$

$P \leftarrow 0$

$T \leftarrow V$

$d(s, v) = \infty$ per ogni $v \in V$, $d(s, s) = 0$, $pred(s) = 0$

while $P \neq V$ **do**

$v = argmin\{d(s, v) | v \in T\}$

$P := P \cup v, T := T \setminus v$

for $w \in N(v)$ **do**

if $d(s, w) > d(s, v) + \omega(v, w)$ **then**

$d(s, w) := d(s, v) + \omega(v.w)$

$pred(w) = v$

end if

end for

end while

1.4.2 All-Pair Shortest Path

Per il problema APSP, l'obiettivo è quello di trovare il cammino minimo tra ogni coppia di nodi nel grafo. Questo problema può essere risolto, ad esempio, tramite la soluzione del problema SSSP per ogni nodo del grafo come nodo sorgente, per poi combinare i risultati per ottenere il cammino più corto tra ogni coppia di nodi. A tal fine, è possibile applicare l'algoritmo di Dijkstra (1) n volte, utilizzando ad ogni iterazione un vertice diverso per s. Questo approccio è efficiente per grafi sparsi, in particolare per grafi non pesati ha una complessità temporale $\mathcal{O}(nm + n^2)$. Per grafi densi e pesati ($m \in \Omega(n^2)$) tuttavia questo approccio non è efficiente, dato che la complessità temporale è $\mathcal{O}(n(m + n \log n))$.
Indtroduciamo quindi un nuovo algoritmo che migliora le prestazioni basandosi sulla seguente osservazione.

Lemma 1. *Siano $d(u, v)$ $u, v \in V$ le lunghezze di un qualche cammino da u a v. Quelle lunghezze rappresentano un distanza geodesica se e solo se $d(u, w) \leq d(u, v) + d(v, w)$ per ogni $u, v, w \in V$.*

L'algoritmo di Floyd-Warshall[5] risolve il problema di tutti i cammini minimi (APSP) in un grafo orientato o non orientato con pesi sugli archi positivi o negativi. La sua

complessità temporale è $\mathcal{O}(n^3)$. Questo algoritmo in sostanza controlla se esiste una tripla di nodi u, v, w tali che il Lemma 1 non sia rispettato e in quel caso corregge l'etichetta con il nuovo valore $d(u, w)$.

Algorithm 2 Algoritmo di Floyd-Warshall per APSP

Input: Grafo $G = (V, E)$, pesi archi $\omega : E \longrightarrow \mathbb{R}$

Output: Distanze geodesiche $d(u, v)$ per ogni $u, v \in V$

```
for all u, v ∈ V do
    d(u, v) = ∞, pred(u, v) = 0
end for
d(v, v) = 0 per ogni v ∈ V
for all u, v ∈ V do
    d(u, v) = ω(u, v), pred(u, v) = u
end for
for v ∈ V do
    for u, w ∈ V × V do
        if d(u, w) > d(u, v) + d(v, w) then
            d(u, w) := d(u, v) + d(v, w)
            pred(u, w) := pred(v, w)
        end if
    end for
end for
```

1.4.3 Approccio ingenuo per il calcolo della betweenness centrality

In questa Sezione introduciamo un approccio ingenuo che valuta l'Equazione 1.5, ne calcola i termini in maniera diretta e restituisce il valore di BC per ogni nodo senza effettuare alcun tipo di ottimizzazione. E' in assoluto l'agolritmo più semplice da implementare in quando non fa altro che applicare l'equazione 1.5.

Questo algoritmo funziona calcolando i percorsi piú brevi tra tutte le coppie di nodi nel grafo risolvendo il problema APSP, e poi contando il numero di volte in cui ciascun nodo appare su un cammino minimo tra tutte le coppie di nodi. L'algoritmo si divide in quattro parti.

1. Inizializza $b(v) = 0$ per ogni nodo $v \in V$

2. Risolve il problema APSP per il grafo, quindi trova tutti i cammini minimi per ogni coppia di nodi del grafo e li salva.

3. Per ogni coppia di nodi s e t conta il numero di volte che v compare nei cammini, ottenendo quindi il valore $\sigma_{st}(v)$ dell'equazione 1.5.

4. Divide $\sigma_{st}(v)$ per il numero totale di cammini minimi tra s e t, σ_{st} e aggiorna $b(v)$.

L'algoritmo ingenuo descritto negli step precedenti calcola la betweenness centrality di tutti i nodi di un grafo con una complessità temporale di $\mathcal{O}(n^3)$, dove n è il numero di nodi nel grafo.[5]

1.4.4 Algoritmo di Brandes

Brandes introduce un algoritmo efficiente per il calcolo esatto della BC con complessità temporale $\mathcal{O}(nm + n^2 \log(n))$ per grafi pesati ($\mathcal{O}(nm)$ per grafi non pesati) e complessità spaziale $\mathcal{O}(n + m)$.[3]
Andiamo a dare alcune definizioni utili per capire il funzionamento dell'algoritmo:

- La dipendenza di coppia (pair-wise dependency) è il rapporto tra i cammini minimi tra s e t che passano per v e il numero totale di cammini minimi tra s e t, ovvero

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{1.7}$$

- La dipendenza unilaterale (one-sided dependency) è definita come la somma delle frazioni di cammini minimi che passano attraverso un nodo v, considerando solo i nodi che precedono v in ciascun cammino minimo, ovvero

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v). \tag{1.8}$$

- L'insieme $P_s(v)$ dei predecessori di un nodo v lungo i cammini minimi da s è definito da

$$P_s(v) = \{u \in V : \{u, v\} \in E, d(s, v) = d(s, u) + \omega(u, v)\}, \tag{1.9}$$

  dove E è l'insieme degli archi, $\omega(u, v)$ è il peso dell'arco $\{u, v\}$ e $d(n_1, n_2)$ rappresenta la distanza geodesica.

Da 1.7 e 1.8 otteniamo che

$$b(v) = \sum_{s,t \in V} \delta_{st}(v) = \sum_{s \in V} \delta_{s\bullet}(v) \tag{1.10}$$

L'algoritmo di Brandes fa uso di una proprietà di $\delta_{s\bullet}(v)$, ovvero che questo valore può essere calcolato ricorsivamente a partire dai valori di $\delta_{s\bullet}(w)$ per i nodi successori di v lungo i cammini minimi che partono da s.

In particolare, assumendo che esista solamente un cammino minimo da s a ogni nodo t abbiamo che, o v giace sull'unico cammino minimo tra s e t e quindi $\delta_{st}(v) = 1$, oppure v non giace sul cammino e in quel caso $\delta_{st}(v) = 0$.
Inoltre v giace su tutti i cammini minimi verso quei nodi per i quali v è predecessore.
Otteniamo quindi la relazione seguente (che vale solo se esiste esattamente un solo cammino minimi tra s e ogni nodo t del grafo).

$$\delta_{s\bullet}(v) = \sum_{w:v\in P_s(w)} (1 + \delta_{s\bullet}(w)). \tag{1.11}$$

Nel caso generale in cui esista più di un cammino minimo tra s e t si ha che

$$\delta_{s\bullet}(v) = \sum_{w:v\in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)). \tag{1.12}$$

Grazie a questa proprietà siamo in grado di calcolare la BC risolvendo un SSSP per ogni nodo del grafo e andando a sommare ad ogni iterazione la one-sided dependency del nodo s su ogni altro nodo v e aggiungerlo a $b(v)$.

Capitolo 2

Approssimazione della betweenness centrality

2.1 Introduzione

Le sezioni precedenti hanno introdotto algoritmi in grado di calcolare in maniera esatta il valore della betweenness centrality. L'algoritmo di Brandes in particolare offre un metodo efficace di calcolo della BC, tuttavia la sua complessità temporale lo rende inutilizzabile per l'analisi di grafi molto grandi in quanto i costi in termini di tempo diventerebbero proibitivi. Diverse euristiche sono state implementate per migliorare l'algoritmo di Brandes, tuttavia la complessità al caso pessimo rimane invariata. Bisogna quindi introdurre dei nuovi algoritmi in grado di fornire stime dei valori di BC con prestazioni migliori. Questi algoritmi si concentrano sul calcolo di un'approsimazione rigorosa della BC, sfruttando un migliore trade-off tra l'accuratezza e il tempo richiesto. La perdita di precisione, finché rientra in limiti di tolleranza accettabili, non costituisce un problema, in quanto molto spesso non siamo interessati agli esatti valori di BC dei nodi quanto piuttosto al loro valore relativo. Affinché questi algoritmi abbiamo una qualche utilità pratica devono quindi scalare bene e avere una bassa dipendenza del runtime rispetto alla dimensione della rete (numero di nodi e archi). Definiamo questa nuova tipologia di algoritmi come algoritmi di aprossimazione e vediamo più nel dettaglio come funzionano e come ottengono tali specifiche sopra citate.

2.2 Algoritmo di approssimazione basato su random sampling

Introduciamo un algoritmo di algoritmo di approssimazione della BC basato sul campionamento randomico di k coppie di nodi. Lo pseudocodice è il seguente.

Algorithm 3 Algoritmo di approssimazione per la BC

Input grafo (pesato/non pesato) $G = (V, E)$ con $n = |V|$, $m = |E|$, $k \in \mathbb{N}$

Output approssimazione $\hat{b}(v)$ di $b(v)$ per ogni $v \in V$

```
for all v ∈ V do
    b̂(v) ← 0
end for
for i ← 1 a k do
    (s, t) ← coppia di vertici presi uniformemente a caso da V con s ≠ t
    P_st ← insieme di tutti i cammini minimi da s a t
    π_i ← cammino minimo scelto uniformemente a caso da P_st
    for all v ∈ π_i do
        if v ≠ s e v ≠ t then
            b̂(v) ← b̂(v) + 1/k
        end if
    end for
end for
return b̂
```

Questo algoritmo inizializza i valori di $\hat{b}(v)$ a zero e poi esegue k iterazioni. In ogni iterazione viene selezionata una coppia di nodi uniformemente a caso dal grafo, calcolato l'insieme P_{st} di tutti i cammini minimi tra i due nodi s e t e selezionato a caso uno tra questi, salvandolo in π_i. Infine per ogni nodo v interno al cammino minimo π_i (quindi esclusi il nodo s e il nodo t), l'algoritmo aggiorna la stima $\hat{b}(v)$ della betweenness centrality $b(v)$ incrementandola di $\frac{1}{k}$. Al termine, l'algoritmo ritorna il vettore $\hat{b}$ che contiene tutti valori $\hat{b}(v)$ per ogni $v \in V$.

L'insieme di tutti i cammini minimi P_{st} può essere calcolato da una versione modificata dell'algoritmo SSSP, che inizia dal nodo s e si ferma quando tutti i cammini minimi verso t sono stati trovati. L'esecuzione di questo algoritmo a partire da un numero di vertici inferiore a n permette di ottenere queste stime in tempo considerevolmente ridotto.

Notiamo che la qualità dell'approssimazione dipende dal parametro k, infatti più k è grande e più l'approssimazione $\hat{b}(v)$ si avvicina al valore esatto $b(v)$. Allo stesso tempo, un valore molto grande di k implica dover compiere molte iterazioni, ciascuna potenzialmente costosa in termini computazionali. Questo rappresenta un significativo trade-off tra il tempo richiesto per il calcolo delle stime e l'accuratezza delle stesse. Un problema importante è quindi trovare il minimo numero di campioni k che ci garantisca una sufficiente accuratezza di $\hat{b}(v)$ rispetto al valore esatto $b(v)$ minimizzando quindi la complessità temporale dell'algoritmo.

Una proprietà molto importante di questo algoritmo è il fatto di essere uno stimatore unbiased, ovvero che il valore atteso $E[\hat{b}(v)]$ di $\hat{b}(v)$ è uguale a $b(v)$

$$\mathbb{E}[\hat{b}(v)] = b(v), \quad \forall v \in V. \tag{2.1}$$

Dimostrazione 1. *Sia X_i, $1 \leq i \leq k$ una variabile aleatoria con valore pari a 1 se $v \in \pi_i$ e 0 altrimenti.*
Allora si può definire $\hat{b}(v)$ come

$$\hat{b}(v) = \frac{1}{k} \sum_{i=1}^{k} X_i. \tag{2.2}$$

Quindi, il valore atteso di $\hat{b}(v)$ è uguale a

$$\mathbb{E}[\hat{b}(v)] = \mathbb{E}[\frac{1}{k} \sum_{i=1}^{k} X_i], \tag{2.3}$$

e per linearità del valore atteso vale che

$$\mathbb{E}[\frac{1}{k} \sum_{i=1}^{k} X_i] = \frac{1}{k} \sum_{i=1}^{k} \mathbb{E}[X_i]. \tag{2.4}$$

Dato che X_i è una variabile aleatoria di bernoulli, abbiamo che

$$\mathbb{E}[X_i] = P([X_i = 1]) = P(v \in \pi_i). \tag{2.5}$$

Per il teorema della probabilità totale

$$P(v \in \pi_i) = \sum_{s \neq v \neq t} P(v \in \pi_i | s, t \text{ sono selezionati}) P(s, t \text{ sono selezionati}). \tag{2.6}$$

Osserviamo che

$$P(s, t \text{ sono selezionati}) = \frac{1}{n(n-1)}, \tag{2.7}$$

e che

$$\sum_{s \neq v \neq t} P(v \in \pi_i | s, t \text{ sono selezionati}) = \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{2.8}$$

Quindi possiamo concludere che

$$\hat{b}(v) = \frac{1}{k} \sum_{i=1}^{k} X_i = \frac{1}{n(n-1)} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} = b(v). \tag{2.9}$$

□

Definiamo quindi in maniera precisa la tipologia di approssimazione che siamo interessati a ottenere.

Definizione 1 (ϵ-approximation)**.** *Dato $\epsilon \in (0, 1)$, una ϵ-approximation su B è una collezione*

$$\hat{b} = \{\hat{b}(w), w \in V\} \tag{2.10}$$

tale che, per ogni $w \in V$,

$$|\hat{b}(w) - b(w)| \leq \epsilon \tag{2.11}$$

Il seguente risultato permette di ottenere un limite al numero di campioni per ottenere una ϵ-approximation.
Si definisce $VD(G)$ (vertex diameter di G) il massimo numero di vertici interni in un cammino minimo di G.

Teorema 1. *Dato ϵ, parametro di precisione di approssimazione, e δ, parametro di confidenza di approssimazione, se*

$$k \geq \frac{2}{\epsilon^2}(\lfloor \log_2(VD(G) - 2) \rfloor + \ln(\frac{1}{\delta})) \tag{2.12}$$

allora

$$Pr([\exists v \in V \text{ tale che } |\hat{b}(v) - b(v)| > \epsilon]) < \delta. \tag{2.13}$$

Questo teorema in pratica ci dice che se vogliamo che tutte le stime $\hat{b}(v)$ stiano entro un errore ϵ dal valore esatto $b(v)$ con una probabilità di almeno $1 - \delta$ allora l'algoritmo ha bisogno di $\mathcal{O}(\frac{\log(\frac{VD(G)}{\delta})}{\epsilon^2})$ campioni.[9]

2.3 SILVAN

SILVAN è un algoritmo di approssimazione della BC basato sulla tecnica del progressive sampling sviluppato da Fabio Vandin e Leonardo Pellegrina[8].

Una sua implementazione è disponibile online [1].
L'algoritmo, in maniera simile a quanto discusso in precedenza, utilizza il campionamento uniforme di cammini minimi nel grafo e utilizza la frazione di cammini minimi che passano per v come uno stimatore unbiased della BC di v per ottenere una ϵ-approximation di $b(v)$ per ogni $v \in V$. Rispetto però all'algoritmo 3 introdotto in precedenza, SILVAN utilizza una strategia diffrente.

SILVAN introduce un nuovo stimatore il quale, presi due nodi uniformemente a random nel grafo va a scegliere un insieme di cammini minimi tra i due nodi.

SILVAN campiona i cammini minimi nei seguenti 3 passaggi:

1. sceglie due nodi, s e t uniformemente a caso;
2. esegue una BFS bidirezionale partendo dai nodi s e t, fermandosi quando le due BFS si incontrano;
3. campiona $\lceil \alpha\sigma_{st} \rceil$ cammini minimi uniformemente a caso dall'insieme Π_{st} di cammini minimi tra s e t dove $\sigma_{st} = |\Pi_{st}|$ è il numero totale di cammini minimi tra s e t e α è un parametro costante maggiore o uguale a 1.

Grazie alla BFS bidirezionale, introdotta da [2], è possibile ottenere tutte le informazioni necessarie a campionare un cammino minimo tra s e t in tempo $\mathcal{O}(|E^{\frac{1}{2}}|)$.
Con questa procedura è possibile verificare che la BC di v è pari alla frazione di cammini minimi campionati, quindi che anche lo stimatore utilizzato da SILVAN è unbiased [8].

Un'altra differenza respetto all'algoritmo precedente, riguarda l'utilizzo di una strategia di progressive sampling. Al posto di considerare k campioni, con k fornito in input all'algoritmo, SILVAN analizza il grafo in maniera adattiva, potenzialmente terminando dopo aver considerato un minor numero di campioni rispetto a quanto indicato dal Teorema 1. Ad alto livello, SILVAN è un algoritmo di progressive sampling che funziona attraverso processi iterativi.
All'iterazione i-esima l'algoritmo estrae un'approssimazione della betweenness centrality per tutti i nodi del grafo da una collezione S_i di $s_i = |S_i|$ campioni randomici da un certo

[1] https://github.com/VandinLab/SILVAN

dominio D (nel nostro caso, i campioni sono insiemi di cammini minimi tra coppie di nodi). L'algoritmo al termine della fase i, controlla se la collezione S_i permette di ottenere una ϵ-approximation, ovvero se viene soddisfatta una condizione di arresto basata sulle informazioni ottenute dai campioni S_i e dall'approssimazione calcolata. Se la condizione non è soddisfatta, l'algoritmo crea una nuova collezione S_{i+1} a partire dalla collezione S_i e aggiungendo nuovi campioni estratti uniformemente a caso fino a raggiungere dimensione s_{i+1}. Infine calcola una nuova approssimazione dalla nuova collezione S_{i+1} e controlla nuovamente la condizione di arresto, e così via.
I valori delle taglie S_i vengono fissati da una schedule che permette di verificare la ϵ-approximation su taglie diverse, calcolate in maniera automatica sulla base delle caratteristiche del grafo. Se invece la condizione è soddisfatta, significa che l'approssimazione trovata ha la qualità desiderata dell'ϵ-approximation ($|\hat{b}(v) - b(v)| < \epsilon$) per ogni v con una probabilità di almeno $1 - \delta$.
Una buona condizione di arresto segue questi principi:

1. deve essere valutata efficaciemente.

2. deve essere soddisfatta per piccole dimensioni del campione.

3. se soddisfatta, deve garantire che l'approssimazione sia, con una probabilità almeno $1 - \delta$, una ϵ-approximation.

Un'ulteriore differenza rispetto all'algoritmo 3 introdotto in precedenza riguarda le tecniche probabilistiche utlizzate da SILVAN al fine di garantire una ϵ-approximation. Al posto di sfruttare il Teorema 1, il quale può fornire delle garanzie lasche sull'approsimazione, SILVAN utilizza le Rademacher Averages, una tecnica avanzata di statistical learning per l'analisi della convergenza probabilistica delle medie di funzioni.[6]

Capitolo 3

Valutazione sperimentale

In questa sezione presentiamo una valutazione sperimentale dell'algoritmo di Brandes e dell'algoritmo SILVAN. In particolare andremo a misurare i tempi di esecuzione per entrambi, inoltre per SILVAN andremo ad analizzare anche il numero di cammini minimi campionati in funzione di ϵ, e analizzeremo l'impatto che il parmetro α introdotto in Sezione 2.3 ha sul tempo di esecuzione e sul numero di cammini minimi campionati.

3.1 Ambiente sperimentale

Gli esperimenti sono stati condotti sul cluster "turing" del dipartimento di Ing. dell'informazione dell'Università di Padova. Le specifiche della macchina sono le seguenti:

- sistema operativo Ubuntu 20.4
- 72 core Intel Xeon 2.30 Ghz CPU
- 1 TB RAM

Gli esperimenti sono stati eseguiti da remoto collegandosi al cluster attraverso protocollo SSH ed avviando un ambiente virtuale python.

3.2 Strumenti utilizzati

Gli esperimenti sono stati eseguiti in ambiente python attraverso uno script disponibile online [1]. Le librerie utilizzate sono le seguenti:

[1] https://github.com/Apenzz

- pandas: è una libreria open-source specializzata nell'elaborazione e manipolazione di dati strutturati, come ad esempio i dati tabulari. Essa fornisce una vasta gamma di funzioni per la lettura, la scrittura e la gestione dei dati, tra cui la manipolazione delle serie e dei dataframe, la selezione, il filtraggio e l'ordinamento dei dati, l'aggregazione e la trasformazione dei dati.

- numpy: è una libreria open-source specializzata nell'elaborazione di array multidimensionali e matrici. Essa fornisce diverse funzioni matematiche e statistiche, tra cui l'ordinamento, la selezione e la manipolazione dei dati, la generazione di numeri casuali, l'elaborazione delle immagini e la trasformata di Fourier.

- matplotlib: è una libreria open-source specializzata nella creazione di grafici e visualizzazioni. Viene utilizzata per la creazione di grafici 2D e 3D, tra cui la personalizzazione dell'aspetto del grafico, l'aggiunta di titoli, etichette e legende, la creazione di sottografici e l'utilizzo di diverse tipologie di grafici, come ad esempio linee, barre, istogrammi, e scatter plot.

- os: è una libreria di python facente parte della libreria standard che offre diverse funzioni per l'interazione con il sistema operativo. Essa consente di accedere a informazioni sul sistema, come ad esempio le variabili d'ambiente, il percorso di lavoro corrente e il nome dell'utente. Inoltre, consente di creare, eliminare e rinominare file e directory, di accedere alle informazioni sui file, come ad esempio le proprietà del file e le date di creazione e modifica. Essa inoltre fornisce anche funzionalità per l'esecuzione di comandi del sistema operativo, la gestione dei percorsi di sistema, la manipolazione delle stringhe di percorso e altro ancora.

- networkit: è una libreria open-souce specializzata nell'analisi di reti su larga scala mantenendo come focus la scalabilità e la parallelizzazione degli algoritmi che implementa. Essa fornisce diverse funzioni per il calcolo di misure, tra le quali metriche di centralita', coeffiencienti di clustering e distribuzione dei gradi dei nodi.

Per il trasferimento dei file tra client e remoto si è utilizzato il programma FileZilla [2].

[2]https://filezilla-project.org/

3.3 Dati sperimentali

L'obbiettivo principale degli esperimenti è quello di misurare le performance dell'algortimo SILVAN su grafi di diverse dimensioni. Per lo studio delle performance abbiamo scelto 6 grafi, 3 diretti e 3 non diretti. Le caratteristiche di questi grafi sono descritte in Sezione 3.3.1.

3.3.1 Grafi analizzati

Di seguito riportiamo i grafi analizzati fornendo una breve descrizione di cosa rappresentano. Le tabelle riportano alcune proprietà di questi grafi, ovvero numero di nodi e archi, numero di nodi e archi nelle componenti connesse piú grandi (CDC sta per componente debolmente connessa, CFC sta per componente fortemente connessa) e diametro (lunghezza del cammino minimo piú lungo tra due nodi del grafo).

Rete di email Enron La rete di email Enron copre tutte le comunicazioni email di un dataset di circa mezzo milione di email. I nodi della rete sono gli indirizzi email e se un indirizzo ha inviato almeno una email ad un altro allora la rete contiene un arco non orientato tra di loro.

Nodi	36692
Archi	183831
Nodi in CDC piú grande	33696
Archi in CDC piú grande	180811
Nodi in CFC piú grande	33696
Archi in CFC piú grande	180811
Diametro	11

Rete di prodotti Amazon comprati assieme Il grafo si basa sulla funzionalità "Clienti che hanno acquistato questo articolo hanno acquistato anche" del sito Amazon. Se un prodotto è spesso acquistato con un altro prodotto, viene creato un collegamento tra i due prodotti nella rete.

Nodi	334863
Archi	925872
Nodi in CDC piú grande	334863
Archi in CDC piú grande	925872
Nodi in CFC piú grande	334863
Archi in CFC piú grande	925872
Diametro	44

Rete di collaborazione DBLP Il database bibliografico DBLP fornisce un elenco completo di articoli di ricerca in informatica. La rete è composta di co-autori in cui due autori sono connessi se pubblicano almeno un articolo insieme.

Nodi	317080
Archi	1049866
Nodi in CDC piú grande	317080
Archi in CDC piú grande	1049866
Nodi in CFC piú grande	317080
Archi in CFC piú grande	1049866
Diametro	21

Rete di votazioni Wikipedia La rete contiene tutti i dati di voto dati dagli utenti Wikipedia per eleggere gli amministratori (admin) di Wikipedia dall'inizio di Wikipedia fino a gennaio 2008. La rete è costituita da nodi che rappresentano gli utenti di Wikipedia, mentre un arco diretto dal nodo i al nodo j rappresenta il fatto che l'utente i ha votato per l'utente j.

Nodi	7115
Archi	103689
Nodi in CDC piú grande	7066
Archi in CDC piú grande	103663
Nodi in CFC piú grande	1300
Archi in CFC piú grande	39456
Diametro	7

Rete di citazioni HEP-PH Arxiv HEP-PH è un archivio online di articoli scientifici di fisica ad alta energia (high-energy physics) Il grafo tratto dall'e-print arXiv copre tutte le citazioni all'interno del periodo gennaio 1993 - aprile 2003.
Il grafo contiene solo le citazioni tra gli articoli dell'archivio e non include informazioni sugli articoli citati al di fuori di esso.

Nodi	34546
Archi	421578
Nodi in CDC piú grande	34401
Archi in CDC piú grande	421485
Nodi in CFC piú grande	12711
Archi in CFC piú grande	139981
Diametro	12

Rete web Google I nodi del grafo rappresentano le pagine web e gli archi diretti rappresentano i collegamenti ipertestuali tra di esse. Questo grafo è stato rilasciato da Google come parte del Google Programming Contest nel 2002.

Nodi	875713
Archi	5105039
Nodi in CDC piú grande	855802
Archi in CDC piú grande	5066842
Nodi in CFC piú grande	434818
Archi in CFC piú grande	3419124
Diametro	21

3.3.2 Fase preliminare

I grafi sono stati scaricati da una repository disponibile online [3] e in seguito sono stati pre-processati con uno script il quale legge i grafi e li converte in un nuovo formato in cui gli identificatori dei nodi sono riassegnati in modo che siano consecutivi. Questa riassegnazione degli identificatori dei nodi è necessaria perché alcuni algoritmi di analisi dei grafi richiedono che gli identificatori dei nodi siano numerati consecutivamente.

[3] https://snap.stanford.edu/data/

3.4 Valutazione sperimentale dell'algoritmo di Brandes

Questo primo esperimento considera le prestazioni temporali dell'algoritmo di Brandes, il quale calcola in modo esatto la BC per tutti i vertici del grafo. Di seguito riportiamo la tabella con i tempi di esecuzione medi per i vari grafi. I tempi sono una media calcolata su 10 run dell'algortimo per ogni grafo.

wiki-Vote	0.304153 s
cit-HepPh	5.673113 s
email-Enron	17.88543 s
com-dblp	$\geq$ 1 h
com-amazon	$\geq$ 1 h
web-Google	$\geq$ 1 h

Per i grafi com-dblp, com-amazon e web-Google è stato necessario interrompere forzatamente l'esecuzione e quindi la tabella riporta il tempo di arresto. In conclusione l'algoritmo di Brandes è un algoritmo molto efficiente per calcolare la betweenness centrality di un grafo, ma diventa troppo costoso computazionalmente per grafi di grandi dimensioni. Tuttavia, se l'obiettivo principale è la precisione del valore della betweenness centrality e non la velocità di calcolo, l'algoritmo di Brandes può essere utile, a condizione che il grafo da analizzare sia di piccole dimensioni.

3.5 Valutazione sperimentale di SILVAN

Questa Sezione riporta i risultati di SILVAN. Per tutti gli esperimenti abbiamo utilizzato $\delta = 0.05$, in modo tale da ottenere approssimazioni calcolate da SILVAN con probabilità almeno $1 - \delta$. Tutti gli altri parametri sono stati impostati di default.

3.5.1 Risultati per i tempi di esecuzione vs ϵ

In questo esperimento abbiamo valutato il tempo di esecuzione in funzione di ϵ, il quale controlla l'accuratezza dell'approssimazione calcolata da SILVAN. In questo grafico 3.1 abbiamo tracciato le curve epsilon-tempo per i diversi grafi. In ascissa abbiamo riportato

il valore di epsilon ϵ mentre in ordinata abbiamo riportato il tempo di esecuzione dell'algoritmo espresso in secondi. Per l'esperimento si è fatto variare il parametro epsilon (che assume i valori [0.01, 0.005, 0.0025, 0.001]) e per ogni coppia (grafo, epsilon) sono state eseguite 10 run dell'algoritmo. Il tempo riportato nel grafico è quindi una media sulle 10 run. Entrambi gli assi sono mostrati con scala logaritmica. Le barre verticali rappresentano la deviazione standard rispetto al tempo medio. Il parametro alpha è rimasto fisso al valore 2.3 (valore di default) per l'intera durata dell'esperimento.

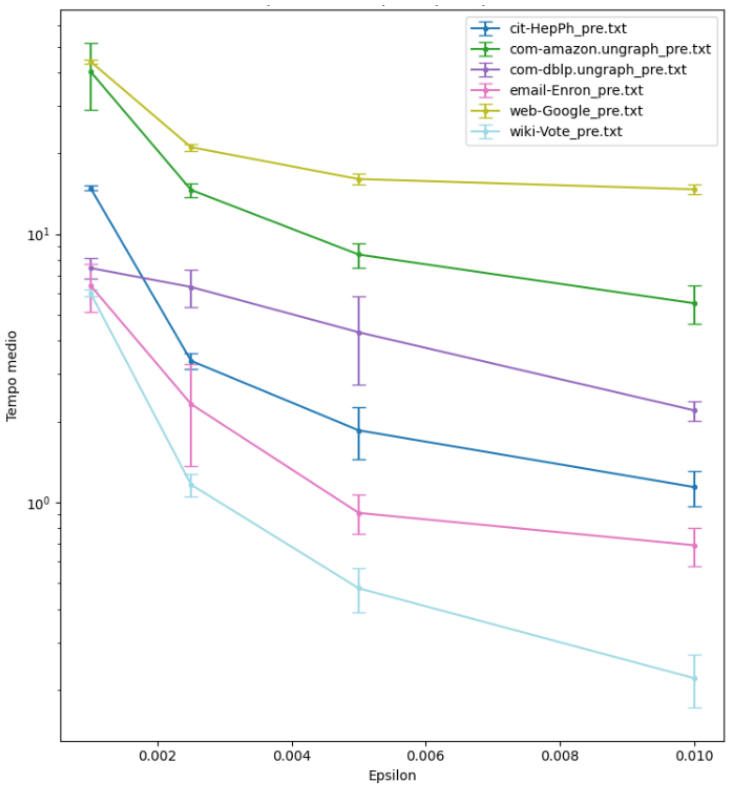


Figura 3.1: Epsilon vs tempo medio per alpha=2.3

Osserviamo che il tempo medio di esecuzione dell'algoritmo cresce rapidamente al diminuire del parametro ϵ, infatti piú ϵ diventa piccolo piú deve aumentare la precisione della stima $\hat{b}(v)$ e ciò richiede piú iterazioni dell'algoritmo. Nonostante ciò anche per grafi molto grandi (vedi la curva di web-Google) e con un valore molto piccolo di ϵ il tempo di esecuzione rimane nell'ordine di grandezza delle decine di secondi, un tempo e una precisione piú che accettabili per la maggior parte delle applicazioni moderne.

3.5.2 Risultati per il numero di campioni vs ϵ

In questo esperimento abbiamo valutato il tempo di esecuzione in funzione di ϵ, il quale controlla l'accuratezza dell'approssimazione calcolata da SILVAN. In questo grafico 3.2

abbiamo tracciato le curve epsilon-m per i diversi grafi. In ascissa abbiamo riportato il valore di epsilon ϵ mentre in ordinata abbiamo riportato il valore 'm' ovvero il numero di campioni considerati da SILVAN. Come per il grafo epsilon-time, si è fatto variare il parametro ϵ (che assume i valori [0.01, 0.005, 0.0025, 0.001]) e per ogni coppia (grafo, ϵ) sono state eseguite 10 run dell'algoritmo. Il valore m riportato nel grafico è quindi una media sulle 10 run. Le barre verticali questa volta rappresentano la deviazione standard rispetto al valore medio di m. Il parametro α è rimasto fisso al valore 2.3 (valore di default).

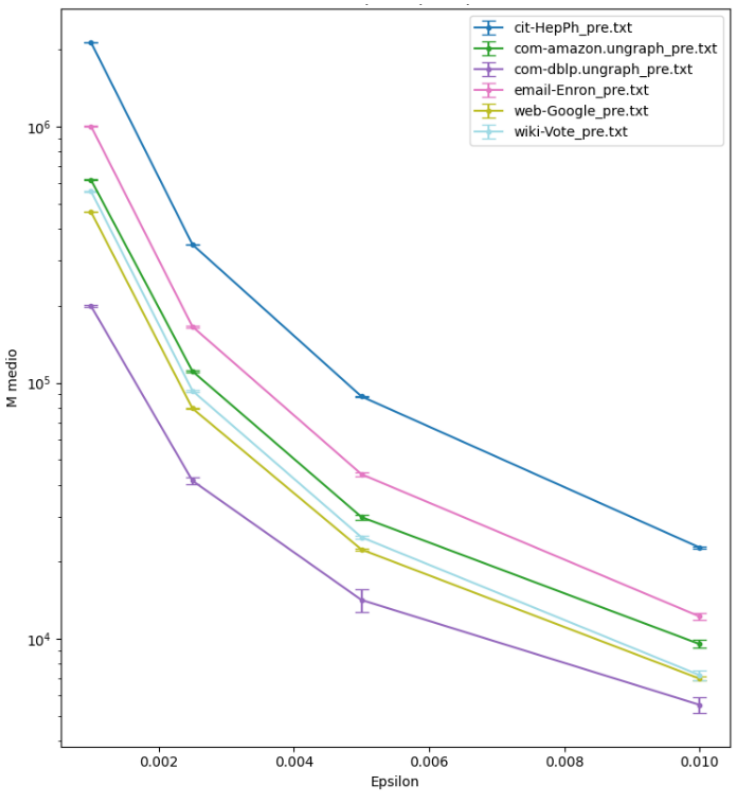
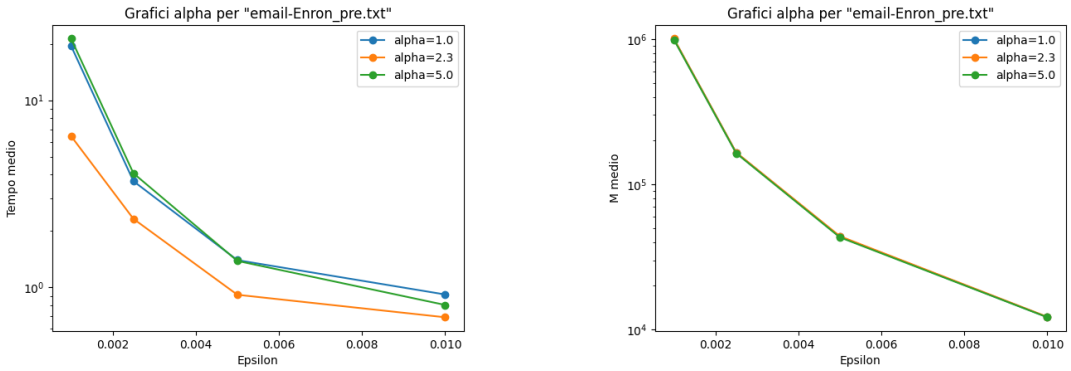


Figura 3.2: ϵ vs m medio per α=2.3

Osserviamo che come per il caso precedente, anche qui, il parametro m è inversamente proporzionale a ϵ quindi per ottenere una precisione maggiore della stima SILVAN ha bisogno di campionare piú cammini minimi. Il risultato è intuitivo in quanto sappiamo che per ottenere il valore esatto della BC dobbiamo campionare tutti i cammini minimi tra le coppie di nodi. Notare che la varianza sul parametro m è molto piú piccola rispetto alla varianza sul tempo del grafico precedente, come indicato dalla lunghezza delle barre verticali.

3.5.3 Impatto del parametro α sulle prestazioni di SILVAN

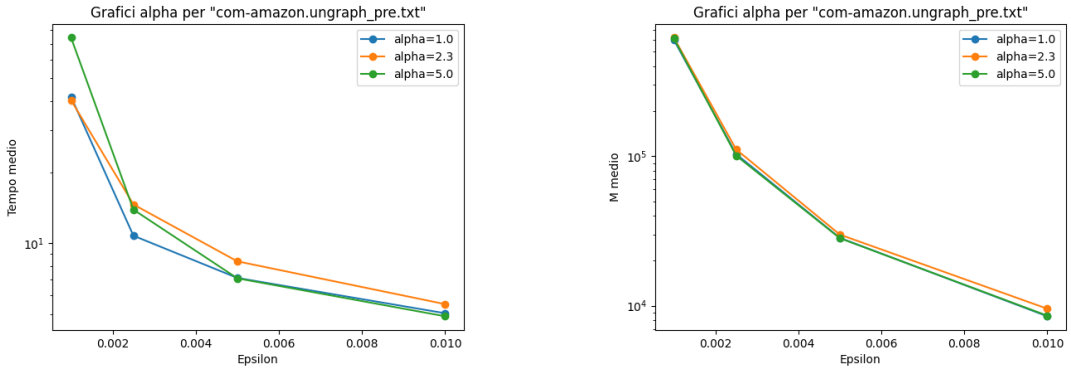
In questo esperimento abbiamo valutato l'impatto del parametro α. A tale fine, abbiamo misurato il tempo di esecuzione e il numero di cammini minimi considerati (m) da SILVAN e riportato la coppia di grafici semilogaritmici epsilon-time e epsilon-m per i diversi grafi andando a tracciare per ogniuno di essi la curva per diversi valori del parametro α. α è un parametro maggiore o uguale a 1 che determina il numero di cammini minimi campionati dall'algoritmo secondo la relazione $\lceil \alpha\sigma_{st} \rceil$ dove σ_{st} è il numero totale di cammini minimi tra i nodi s e t (vedi Sezione 2.3). In questo caso per ogni grafo abbiamo eseguito SILVAN per tre valori diversi di alpha ([1, 2.3, 5]). Per ogni coppia (grafo, epsilon) sono state eseguite ancora una volta 10 run per cui i valori in ordinata sono valori medi.



(a) Epsilon vs time

(b) Epsilon vs m

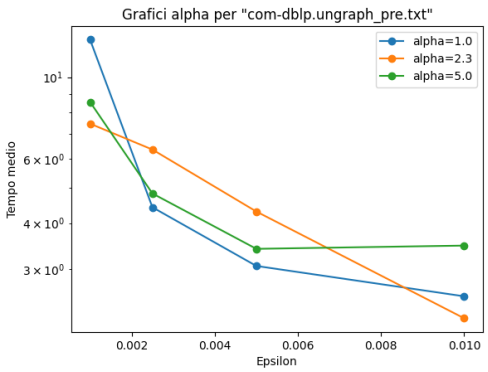
Figura 3.3: Rete di email Enron



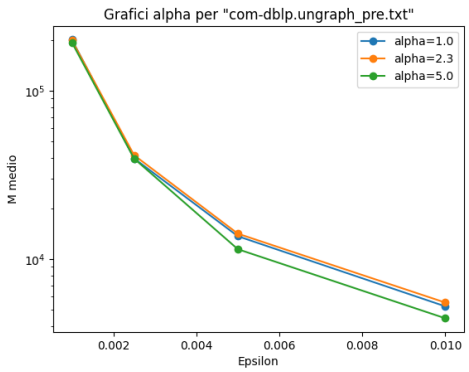
(a) Epsilon vs time

(b) Epsilon vs m

Figura 3.4: Rete di prodotti Amazon

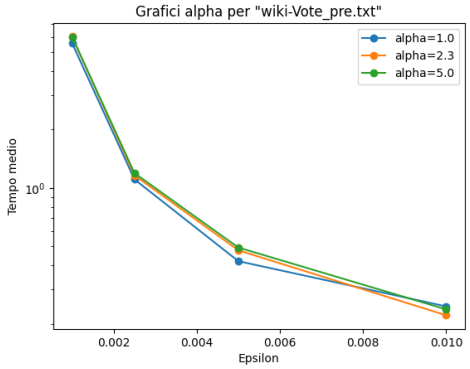


(a) Epsilon vs time

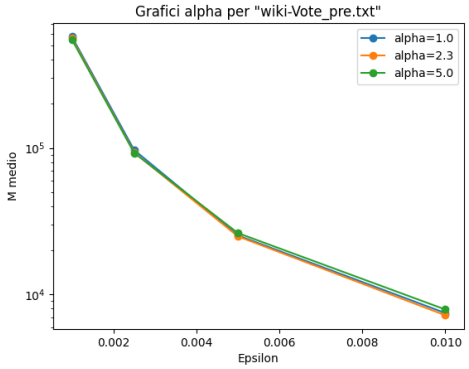


(b) Epsilon vs m

Figura 3.5: Rete di collaborazione DBLP

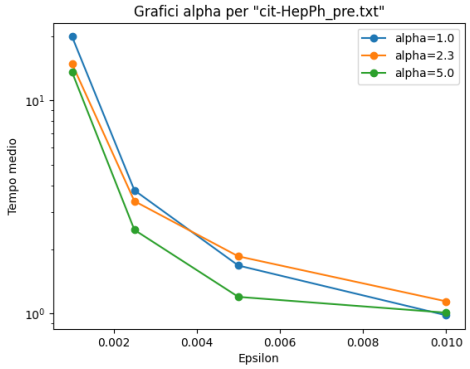


(a) Epsilon vs time

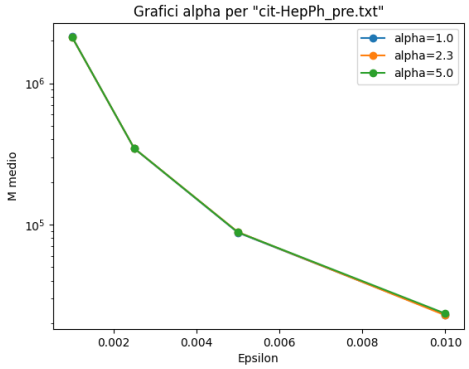


(b) Epsilon vs m

Figura 3.6: Rete di votazioni Wikipedia

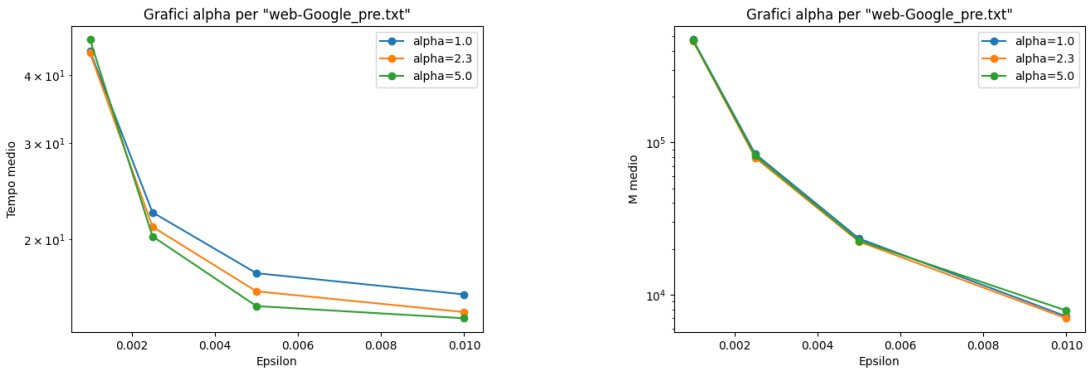


(a) Epsilon vs time



(b) Epsilon vs m

Figura 3.7: Rete di citazioni HEP-PH



(a) Epsilon vs time

(b) Epsilon vs m

Figura 3.8: Rete web Google

Per quanto riguarda i grafici epsilon-tempo notiamo che per ϵ piccolo l'algoritmo sembra performare leggermente meglio per α uguale a 2.3 (ad esempio Fig. 3.5(a)) Per quando riguarda i grafici epsilon-m non ci sono marcate differenze tra le curve, anzi esse sono per lo piú sovapposte e convergono tutte nello stesso punto per ϵ uguale a 0.001. Le uniche differenze osservabili si hanno per valori di epsilon compresi tra [0.005, 0.01] e anche in quel caso sono trascurabili. Dovendo scegliere un valore di α allora 2.3 sembra essere quello migliore.

3.6 Osservazioni

In generale, la scelta dell'algoritmo migliore per calcolare la betweenness centrality dipende dalle esigenze specifiche dell'applicazione e dalle caratteristiche del grafo che si sta analizzando. L'algoritmo di Brandes è risultato piú veloce di SILVAN solamente nell'analisi del grafo wiki-Vote, anche se entrambi terminano in un tempo dell'ordine di pochi secondi. In tutti gli altri casi SILVAN si è dimostrato significativamente piú veloce, anche nel caso piú computazionalmente costoso ovvero quello con ϵ uguale a 0.001. La differenza piú sostanziale si è notata per i grafi di dimensioni piú grandi dove l'algoritmo di Brandes non termina in 1 ora, mentre SILVAN ha finito nel caso peggiore in meno di 50 secondi, diminuendo quindi di quasi due ordini di grandezza il tempo richiesto per ottenere stime accurate e rigorose della BC.

Conclusioni

In conclusione, la betweenness centrality è una misura importante per identificare i nodi più influenti all'interno di una rete. Abbiamo presentato la definizione matematica di questa misura e descritto il calcolo esatto con l'algoritmo di Brandes. Tuttavia, l'algoritmo di Brandes può essere computazionalmente costoso per grafi molto grandi, per cui abbiamo presentato anche un algoritmo approssimato, SILVAN, che può calcolare la betweenness centrality in modo più efficiente.

Siamo poi passati ad esaminare i risultati degli esperimenti su grafi complessi da applicazioni reali, in cui abbiamo confrontato le performance di SILVAN con quelle di Brandes. Dai risultati, è emerso che SILVAN è un algoritmo molto efficace nel calcolo di approssimazioni molto accurate, soprattutto per grafi molto grandi. Valutando l'efficacia di SILVAN per diversi parametri, le sue prestazioni sono rimaste generalmente stabili e ben performanti. In generale, il nostro lavoro dimostra l'efficacia dell'algoritmo SILVAN per il calcolo approssimato di questa misura. I nostri risultati confermano la sua utilità per lo studio e l'analisi di reti e delle loro applicazione in vari campi come la biologia, la sociologia e l'informatica.

Bibliografia

[1] Marc Barthelemy. Betweenness centrality in large complex networks. *The European physical journal B*, 38(2):163–168, 2004.

[2] Michele Borassi and Emanuele Natale. Kadabra is an adaptive algorithm for betweenness via random approximation. *Journal of Experimental Algorithmics (JEA)*, 24:1–35, 2019.

[3] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[4] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 90–100. SIAM, 2008.

[5] Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. *Network analysis: Methodological foundations*, pages 62–82, 2005.

[6] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[7] Mark Newman. Networks: An introduction by m.e.j newman, oxford university press inc., new york, 2010.

[8] Leonardo Pellegrina and Fabio Vandin. Silvan: Estimating betweenness centralities with progressive sampling and non-uniform rademacher bounds. *arXiv preprint arXiv:2106.03462*, 2021.

[9] Matteo Riondato and Evgenios M Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 413–422, 2014.