UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica


# Algoritmi per la Riduzione della Polarizzazione e l'Incremento della Navigabilità nelle Reti Sociali


*Relatore:*
Prof. Leonardo Pellegrina
*Correlatore:*
Prof. Fabio Vandin

*Laureando:*
Francesco Biscaccia Carrara
2008809



Anno Accademico 2022/2023
Data di Laurea 25/09/2023

**Abstract**

La fruizione di contenuti online, con il passare del tempo, è un'attività sempre più presente nella vita quotidiana. Talvolta, però, l'accesso alle informazioni all'interno del Web e delle reti sociali non è completamente libero: molto spesso l'utente rimane intrappolato in una "bolla" che impedisce l'esposizione completa ai contenuti. Tra le varie pagine web e post su reti sociali si forma un forte condizionamento che impone i collegamenti tra argomenti simili, costringendo l'utente ad una visione polarizzata e condizionata delle informazioni. Al fine di trovare una soluzione a questo problema, in questa tesi verranno presentati due algoritmi: RePBubLik e ShuffLik. Entrambe le metodologie permettono di ridurre la polarizzazione e migliorare la navigabilità della rete, ma utilizzando due approcci differenti: RePBubLik permetterà di ridurre il bias strutturale introducendo nuovi collegamenti all'interno della rete, mentre ShuffLik interviene modificandone il peso per aumentare la navigabilità della rete.
Per RePBubLik, inoltre, verrà svolta una valutazione sperimentale delle performance, confrontandolo con approcci alternativi.

# Indice

# Introduzione

Il World Wide Web è stata una delle innovazioni più importanti del ventesimo secolo: nato per aggregare i risultati delle ricerche degli scienziati del CERN, è diventato presto di dominio pubblico e ha permesso a chiunque avesse una connessione Internet di poter ottenere velocemente e facilmente qualsiasi informazione gli servisse. Data questa sua versatilità e semplicità d'uso, il WWW è diventato in un periodo relativamente breve il mezzo più rapido e "completo" per ottenere informazioni. Il volume di nozioni contenute in esso è cresciuto esponenzialmente nel corso degli anni e oggigiorno ogni argomento può essere trovato mediante il WWW. Questa moltitudine di informazioni, però, non garantisce che siano tutte facilmente reperibili dall'utente. Ogni qualvolta si esplora una pagina, infatti, vengono proposti dei link, che a detta dell'autore della pagina, sono pertinenti al contenuto che viene proposto. Questa struttura può introdurre un bias, dettato dai creatori delle web pages, talvolta con finalità malevole: avere molteplici siti che riportano la stessa informazione, non significa necessariamente che quest'ultima sia vera e limitare l'accesso a opinioni diverse può essere sfruttato al fine di manipolare l'opinione degli utenti. Per questo, una corretta esplorazione del Web dovrebbe permettere l'esplorazione di contenuti diversi, talvolta contrastanti, sullo stesso argomento. Così facendo, l'utente può avere un'idea molto più chiara e completa sul topic che sta analizzando.
Questo è l'obiettivo dei due algoritmi proposti in questa tesi: ridurre la propensione delle reti sociali a confinare l'utente in contenuti incompleti o parziali, attraverso l'aggiunta di nuovi collegamenti o mediante lo spostamento di link in posizioni più favorevoli all'interno del sito web.

# Capitolo 1

# Definizioni Preliminari

Per esplicare in modo rigoroso e formale gli algoritmi risolutivi, sono necessarie alcune considerazioni iniziali.

## 1.1 Grafo

Data l'interconnessione tra i vari elementi di cui è costituita, una rete sociale viene spesso rappresentata con un grafo. Un grafo è una struttura matematica costituita da un insieme di vertici corrispondi agli elementi della rete (es. una pagina web o un utente). Ogni vertice è unito ad altri vertici mediante archi. Un arco è una coppia di vertici, con delle caratteristiche. A seconda delle proprietà degli archi, il grafo può essere:

- *Grafo diretto*: ogni arco ha un verso di percorrenza; è una coppia ordinata di vertici.
- *Grafo non diretto*: ogni arco non ha un verso di percorrenza; è una coppia di vertici non ordinata.
- *Grafo semplice*: non ci sono archi multipli (più archi per lo stesso nodo) e self-loop.
- *Grafo pesato*: gli archi hanno un peso, un "costo" di percorrenza dell'arco.

Generalmente, un grafo si denota con $G = (V, E)$ dove $V$ è l'insieme dei vertici ed $E$ è l'insieme degli archi, con i relativi pesi.

### 1.1.1 Grafo delle pagine web

Sia $G = (V, E)$ un grafo diretto e pesato con $|V| = n$ vertici, ognuno dei quali ha sia archi entranti che archi uscenti. L'insieme dei vertici $V$ può essere partizionato in 2 sottoinsiemi $R$ (Red) e $B$ (Blu) tali che $R \cup B = V$ e $R \cap B = \emptyset$. Questi due set saranno necessari per differire una categoria dalle altre, all'interno di una rete sociale. Ad esempio, si può definire come $R$ l'insieme dei link su Wikipedia aventi come topic "Tecnologia" e come $B$ l'insieme di tutti i link di Wikipedia con topic diverso da "Tecnologia". Per ogni $v \in V$, l'insieme dei vertici aventi lo stesso colore di $v$ è denotato con $C_v$ mentre l'insieme

dei vertici di colore differente da $v$ si denota con $\overline{C}_v$.

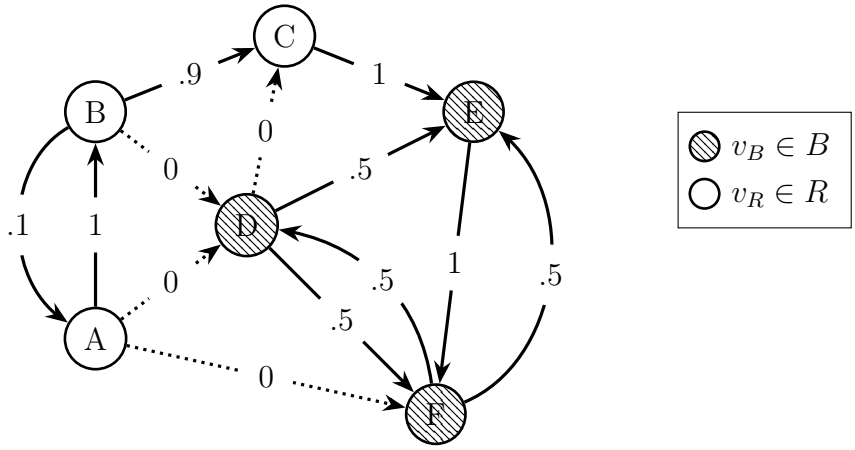



**Figura 1.1:** Esempio di grafo

Ogni arco $(v_i, v_j) \in E$ ha un peso che indica la probabilità di transizione dal vertice $v_i$ al vertice $v_j$. I pesi degli archi vengono raggruppati in una matrice stocastica-destra $M$ $n \times n$ dove ogni elemento $M(v_i, v_j)$ è una probabilità ($0 \leq M(v_i, v_j) \leq 1$). Se $M(v_i, v_j) = 0$ l'arco che collega $v_i$ e $v_j$ non esiste ($(v_i, v_j) \notin E$). Inoltre, per definizione di matrice stocastica-destra, ogni riga di $M$ somma a 1 ($\sum_{v_j=1}^{n} M(v_i, v_j) = 1$).

$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ .1 & 0 & .9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & .5 & .5 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & .5 & .5 & 0 \end{pmatrix} \end{array}$$

**Figura 1.2:** Matrice $M$ per il grafico in Figura 1.1

## 1.2 Random walks

Un *random walk* è un cammino sul grafo che parte da un vertice $v$ e, ad ogni vertice nel cammino, viene scelto, indipendentemente dalle scelte precedenti, un arco uscente con una probabilità uguale al suo peso. Tornerà utile definire una variabile aleatoria $T_v(S)$ che indica il primo istante di tempo in cui un random walk partito dal vertice $v$ raggiunge un vertice di $S \subseteq V$. Si denota con *"hitting time of $S$ from $v$"* la quantità $\mathbb{E}_G[T_v(S)]$ dove l'aspettazione è calcolata tra tutti i possibili random walk che percorrono $G$ partendo da $v$, con le probabilità di transizione date dalla matrice $M$. Nelle realtà, la lunghezza dei random walk è limitata superiormente ad un valore $t$ *("exploration factor")*. Questo fattore è ricavato empiricamente: può essere, ad esempio, il numero di pagine web che un utente visita in una sessione web. Per misurare più accuratamente la lunghezza di una

sessione web, però, si può considerare la variabile aleatoria $T_v^t(S) = \min\{t, T_v(S)\}$.
Dato un grafo $Z$, un qualunque vertice $u$ e un qualunque insieme $S$ di vertici di $Z$, si definisce $u \overset{cond}{\underset{Z}{\rightsquigarrow}} S$ l'evento nel quale un random walk percorre $Z$ da $u$ a un vertice di $S$ senza visitare un vertice di $\overline{C}_u$ (di "colore diverso") e che soddisfa la condizione *cond* in termini di step necessari per raggiungere $S$. Ad esempio, $u \overset{=t}{\underset{Z}{\rightsquigarrow}} S$ indica l'evento nel quale un random walk su $Z$ partendo da $u$ raggiunge un vertice di $S$ con esattamente $t$ steps, senza visitare vertici di $\overline{C}_u$.

## 1.3 Random-walk closeness centrality

Gli algoritmi proposti utilizzano la "Random-walk closeness centrality" come discriminante per la scelta dei nuovi archi da aggiungere o per deterimanre gli archi di cui fare "swapping". Nello specifico, si utilizza la definizione di *Random-walk closeness centrality* per ridurre la lunghezza dei random walk nel grafo in modo che il contributo alla centralità di $v$ dei vertici che non raggiungo $v$ in meno di $t'$ steps (in termini di aspettazione) sia uguale a 0, per ogni $t'$.

**Definizione 1** (Closeness centrality)**.** *La closeness centrality di un vertice è la lunghezza media dei cammini minimi tra un vertice e tutti gli altri vertici del grafo.*

**Definizione 2** (Random-walk closeness centrality (bounded))**.** *Dato un nodo $v \in V$, un sottoinsieme $S \subseteq V$, il t'-bounded Random Walk Closeness Centrality (RWCC) di $v$ rispetto a $S$ è*

$$r^{t'}(v; S) \doteq \frac{1}{|S|} \sum_{w \in S} \Big( t' - \mathbb{E}_G[T_w^{t'}(v)] \Big) \tag{1.1}$$

Dato che il calcolo del RWCC esatto è estremamente costoso, verrà utilizzata come stima $\overline{r}(v)$: si prendono $z$ vertici da $S$ (chiamati $\{w_i\}_{i=1}^z$) ed, eseguendo $k$ random walk, si ottiene la stima $\overline{h}_{w_i}$ di $\mathbb{E}_G[T_{w_i}^{t'}(v)]$ per ogni $w_i$.

$$\overline{r}(v) \doteq t' - \frac{1}{z} \sum_{i=1}^{z} \overline{h}_{w_i} \tag{1.2}$$

## 1.4 Bubble radius

Viene introdotto il concetto di *"Bubble Radius"* per quantificare quanti steps, in media, sono necessari affinchè un utente, partendo con un random walk da un vertice $v \in V$ di un dato colore, raggiunga un vertice di un altro colore.

**Definizione 3** (Bubble radius)**.** *Il Bubble Radius $B_G^t(v)$ di $v$ con $t$ exploration parameter è*

$$B_G^t(v) \doteq \mathbb{E}_G[T_v^t(\overline{C}_v)] \tag{1.3}$$

Data la definizione, è improbabile che un random walk che parte da un vertice $v$, avente un elevato Bubble Radius, raggiunga un vertice di colore opposto ($\in \overline{C}_v$) in meno di/esattamente $t$ steps. Per semplicità, si assume di avere l'esatto valore di BR per ogni vertice del grafo.

### Estensione a più di 2 colori

Nella trattazione degli algoritmi, si assume l'esistenza di solo 2 colori (Red e Blue). Si può estendere il caso a più colori:

- Dividendo il grafo in 2 "colori": il gruppo "Red" conterrà tutti i vertici di un dato colore e il gruppo "Blu" contenente i vertici aventi tutti gli altri possibili colori.
- Assegnando un colore diverso $C_i\,, 1 \leq i \leq k$, per ogni gruppo di vertici.

Nel secondo caso, può essere definito il BR di un vertice $v$ come il minimo tra tutti i $B^t_G(v_i) \doteq \mathbb{E}_G[T^t_{v_i}(C_j)]$ $1 \leq j \leq k, j \neq i$.

# Capitolo 2

# RePBubLik

L'idea alla base nell'algoritmo RePBubLik è quella di aggiungere al grafo della rete sociale archi pesati che permettano di ridurre la polarizzazione rispetto ad un dato topic. Di seguito, vengono esposti i principi teorici e le metriche per valutare l'efficacia di questo approccio.

## 2.1 Vertici cosmopolitan e vertici parochial

In base al valore del Bubble Radius, viene diviso l'insieme dei vertici in 2 sottoinsiemi: *cosmopolitan* e *parochial*. L'insieme $Z(G)$ dei vertici cosmopolitan contiene tutti e soli i vertici di $G$ con un Bubble Radius *al più* uguale a $b$ (cioè $\leq b$), mentre il l'insieme $P(G)$ dei vertici parochial contiene tutti e soli i vertici di $G$ con un Bubble Radius *almeno* uguale a $r$ (cioè $\geq b$), con $1 \leq b < r \leq t$ numeri reali. Importante notare che $Z(G)$ e $P(G)$ sono disgiunti, ma non formano necessariamente una partizione di $V$. Nelle analisi, infatti, partizioneremo il set $P(G)$ a seconda del colore: $P_R(G)$ per i parochial aventi colore $R$ e $P_B(G)$ per i parochial aventi colore $B$. Questa divisione si basa sulla facilità di un nodo di raggiungere, tramite un random walk, un nodo di colore opposto: sarà più facile raggiungere un vertice di colore opposto partendo da vertice cosmopolitan e, al contrario, sarà difficile per un vertice parochial.
Dato il fatto che i nodi parochial sono i vertici per i quali è più difficile trovare un random walk che raggiunga un vertice di colore opposto, risulta sensato utilizzare i valori dei Bubble Radius di quest'ultimi per stimare la polarizzazione (*structural bias*) del grafo.

**Definizione 4** (Structural Bias)**.** *Il Bias Strutturale $\rho(G)$ di $G$ è la somma dei Bubble Radius dei nodi parochial di $G$*

$$\rho(G) = \sum_{v \in P(G)} B_G^t(v) \tag{2.1}$$

## 2.2 Riduzione del bias strutturale mediante inserimento di archi

L'obiettivo di RePBubLik sarà quello di cercare un'insieme di archi, aventi come estremi vertici di colori opposti, da aggiungere al grafo $G$ per diminuire il structural bias. Se, ipoteticamente, si potesse aggiungere una quantità qualsiasi di archi, chiaramente il bias sarebbe 0, in quanto il grafo sarebbe completo. Comprensibilmente, questo non è realistico se si tratta di reti sociali: un browser può consentire solo un certo numero di link per pagina e l'essere umano non è in grado di distinguire un numero elevato di link all'interno della singola web page.
Per queste ragioni, la riduzione del bias strutturale mediante inserimento di archi, diventa un problema di ottimizzazione.

**Definizione 5.** *Dato un set $\Sigma$ di archi non presenti in $G$, si denota con $G_\Sigma$ il nuovo grafo $G_\Sigma \doteq (V, E \cup \Sigma)$.*

**Problema 1.** *Dato $C \in R, B$, trovare il più piccolo insieme $\Sigma$ di vertici distinti $(v, w) \notin E$ con $C_v = C$ e $C_w \neq C$ dove $P_C(G_\Sigma) = \emptyset$.*

In altri termini, il problema da risolvere all'ottimo consiste nel trovare un insieme minimo di archi aventi come estremi vertici di colore opposto, che permetta di azzerare il numero di vertici parochial. Il problema 1 è NP-Hard e APX-Hard, cioè un problema non risolvibile all'ottimo in tempo polinomiale, ma ammette un'approssimazione polinomiale.

### 2.2.1 Approssimazione del problema

Per approssimare il problema 1, saranno necessarie delle nuove metriche per analizzare come cambia il grafo $G$ in seguito all'aggiunta di nuovi archi.

**Definizione 6.** *(Guadagno di $U$ grazie a $\Sigma$) Si assuma di aggiungere al grafo $G$ tutti gli archi di $\Sigma$, ogni edge $e = (v, w)$ ha un peso $M(e)$. Per un set $U$ di vertici, si definisce guadagno di $U$ grazie a $\Sigma$*

$$\Delta(G, U, \Sigma, \{M(e)\}_{e\in\Sigma}, t') \doteq \frac{1}{|U|} \sum_{u\in U} \left( B_G^{t'}(u) - B_{G_\Sigma}^{t'}(u) \right) \tag{2.2}$$

In altri termini, si valuta il cambiamento medio del Bubble Radius dei vertici contenuti in U, cioè quelli coinvolti dagli archi di $\Sigma$. Ogni arco aggiunto, quindi, deve avere un peso $M(e)$. Questo peso è dato da un oracolo che calcola il peso di un arco $M(v, w)$ in funzione di $v$ e delle informazioni *locali* di $v$ (es. numero di archi uscenti). Importante notare che, dopo l'aggiunta di un arco $e = (v, w)$, gli altri archi uscenti dal vertice $v$ avranno un peso scalato di $1 - M(e)$. Date queste considerazioni il problema 1 può essere approssimato come segue.

**Problema 2.** *Sia $C \in \{R, B\}$. Trovare un set $\Sigma = \{(v_i, w_i)\}_{i=1}^{k}$ di $k$ archi con $v_i \in C$ e $w_i \notin C$, per $1 \leq i \leq k$, che massimizzi $\Delta(G, P_C(G), \Sigma, \{M(e)\}_{e\in\Sigma}, t)$.*

RePBubLik è l'algoritmo che approssima il problema 2.

**Definizione 7.** *Per un qualsiasi vertice $v$ e $0 \leq t' \leq t$, sia*

$$F_{t'}(v) \doteq 1 - \mathbb{P}\left(v \underset{G}{\overset{<t'}{\rightsquigarrow}} v\right) \tag{2.3}$$

Questa quantità è uno (cioè la probabilità di avere un random walk da $v$ che torni a $v$ in 0 passi) più la probabilità che un random walk da $v$ finisca in $v$ in meno di $t'$ steps.

**Teorema 1** (Approssimazione di RePBubLik)**.** *Sia $\Sigma$ l'output di RePBubLik e $OPT$ la soluzione ottima al problema 2. Sia $\Delta_\Sigma = \Delta(G, P_C(G), \Sigma, \{M(e)\}_{e\in\Sigma}, t)$. Allora*

$$\Delta(G, P_C(G), OPT, \{M(e)\}_{e\in OPT}, t) \leq \left(2\frac{t}{r}\gamma_{t-2} + 1\right)\left(1 + \frac{1}{e}\right)\Delta_\Sigma \tag{2.4}$$

*dove $\gamma_t \doteq \max_{v\in V} F_{t'}(v)$*

Dunque, RePBubLik fornisce una *approssimazione a fattore costante*, sotto l'assunzione che $\gamma_t$ sia una costante e che $r \geq t/2$.

## 2.3 RePBubLik: definizione intuitiva

Dato il fatto che la funzione obiettivo (guadagno $\Delta$) è *monotona* e *sub-modulare*, la scelta gli archi da aggiungere al grafo $G$ può essere efftuata in maniera greedy. L'assunzione sui pesi che l'oracolo assegna agli archi, inoltre, garantisce che *ogni* vertice di colore diverso da quello di partenza può essere scelto come estremo dell'arco, senza che il peso di quest'ultimo venga modificato. Con queste assunzioni, si semplifica notevolmente il calcolo: invece che cercare le coppie $(v, w)$ da aggiungere a $G$, sarà necessario cercare solamente solamente i *vertici di partenza* da cui escono i nuovi archi.
Per la scelta greedy, un buon candidato $v$ è il vertice che con maggior probabilità viene raggiunto da molti altri vertici in $P_{C_v}(G)$, attraverso random walk brevi. Questa proprietà è espressa dal bounded RWCC $r^{t-2}(v, P_{C_v}(G))$.

## 2.4 RePBubLik: pseudocodice

**Algorithm 1** RePBubLik

**Require:** Grafo $G = (V, E)$, numero $k$ di archi che vogliamo inserire, oracolo $W_G : V \times 2^{V \times V} \to [0, 1]$, $C \in \{R, B\}$
**Ensure:** L'insieme $\Sigma$ di $k$ archi da aggiungere, con relativi pesi.

$\Sigma \leftarrow \emptyset$
**for** $i = 1, \ldots, k$ **do**
  $P \leftarrow computeParochials(G_\Sigma, C)$
  $R \leftarrow computeRWCentrality(P, G_\Sigma)$
  $v_i \leftarrow \arg\max_{v \in P} R(v) \times W_G(v, \Sigma)$
  $u_i \leftarrow arbitrary\ in\ \overline{C}_{v_i}$
  $\Sigma \leftarrow \Sigma \cup \{(v_i, u_i)\}$
**end for**
**return** $\Sigma$

L'algoritmo RePBubLik prende come input un grafo $G$, un numero $k$ di archi da inserire, un oracolo $W$ che fornisce il peso dei nuovi archi e un set $C \in \{R, B\}$ di nodi di un dato colore.
Per prima cosa, viene creato l'insieme $\Sigma$ (inizialmente vuoto) che conterrà gli archi da aggiungere. Successivamente, per $k$ volte: viene calcolato il Bubble Radius di ogni nodo in $C$ nel grafo $G_\Sigma$, ottenuto aggiungendo a $G$ gli archi contenuti in $\Sigma$. Si ottiene, così, un set $P$ di nodi parochial del grafo $G_\Sigma$. In seguito, si calcola il RWCC $r^{t-2}(v, P)$ di ogni nodo $v \in P$ e questi valori vengo salvati in un dizionario $R$. Una volta popolato $R$, viene scelto il nodo $v_i \in P$ avente il valore più alto di $R(v_i) \times W_G(v_i, \Sigma)$ e un qualsiasi nodo $u_i$ di colore $\overline{C}_{v_i}$. Così facendo, si ottiene l' arco $(v_i, u_i)$ da aggiungere all'insieme $\Sigma$. Dopo le $k$ iterazioni, infatti, l'insieme $\Sigma$ sarà popolato da $k$ archi, il cui peso è deciso dall'oracolo.

### 2.4.1 RePBubLik+

Come si può notare dallo pseudocodice, RePBubLik ripete il calcolo del Bubble Radius di *tutti i vertici* e del RWCC di *tutti i vertici in P* ad *ogni iterazioni* del ciclo. Questo, a livello computazionale, è molto oneroso.
Per questo, una pratica alternativa a RePBubLik è una sua versione approssimata (RePBubLik+) che calcola il Bubble Radius e RWCC una *sola volta, prima di entare nel ciclo*. Chiaramente, questa approssimazione non permette di ottenere garanzie rigorose sull'approssimazione calcolata, ma diventa necessaria per diminuire il tempo di esecuzione dell'algoritmo.

# Capitolo 3

# ShuffLik

L'idea alla base nell'algoritmo ShuffLik è quella di scambiare i pesi degli archi aventi lo stesso vertice di partenza, per aumentare la navigabilità della rete. Di seguito, vengono esposti i principi teorici e le metriche per valutare l'efficacia di questo approccio.

## 3.1 Diverse navigability

In molte reti sociali, l'aggiunta di archi è spesso un'operazione invasiva e non sempre possibile. Se si pensa, ad esempio, ai sistemi di raccomandazione (come gli articoli suggeriti di Amazon, etc..) risulta evidente il fatto che il numero di elementi consigliabili è limitato. Per questo motivo, l'approccio di RePBubLik è inefficiente e quindi si dovrà considerare una nuova misura che chiameremo *diverse navigability* che tiene in considerazione i valori di Bubble Radius di *tutti* i vertici della rete.

**Definizione 8** (Diverse navigability)**.** *Dato $S \subseteq V$, la diverse navigability $\xi(S)$ di $S$ è l'opposto del Bubble Radius medio dei vertici in $S$.*

$$\xi(S) \doteq -\frac{1}{|S|} \sum_{v \in S} B_G^t(v) \tag{3.1}$$

*Quando $S = V$, si parla di diverse navigability del grafo $G$; si denota con $\xi(G)$*

## 3.2 Aumento della navigabilità mediante lo scambio delle probabilità di transizione

L'idea alla base di ShuffLik è quella di aumentare la navigabilità di una rete scambiando tra loro i pesi di due archi appartenenti allo stesso vertice (es. pagina web). Questo nella pratica è molto sensato: numerosi studi hanno dimostrato che la probabilità che un certo link venga cliccato è spesso collegata da fattori grafici quali posizione, colore, font, etc. Scambiare, quindi, la posizione dei link all'interno della pagina permetterebbe alla rete di essere più fruibile e navigabile. Per questo, si definisce l'insieme dei *diversifying*

*swaps* come l'insieme delle coppie di archi odi che risultano ragionevoli da scambiare, per aumentare la diverse navigability.

**Definizione 9** (Diversifying swaps)**.** *Per $C \in \{R, B\}$, l'insieme $D_C$ dei diversifying swaps è l'insieme di coppie ordinate di archi, dove per ogni $(e, e')$ vale che:*

1. $(e, e') = ((v, w), (v, u))$ *con* $v, w \in C$ *e* $u \in \overline{C}_v$
2. $M(e) > M(e')$

Sostituire il peso di $e$ con il peso $e'$ della coppia $(e, e') \in D_C$, permette di diminuire il Bubble Radius del vertice di partenza $v$ , aumentando così la diverse navigability $\xi(C_v)$ e, conseguentemente, la $\xi(G)$.
Per quantificare il miglioramento apportato da questo swap, sarà necessaria una qualche misura di guadagno.

**Definizione 10** (Guadagno sul BR di $u$ dato dallo swap di $e$ con $e'$)**.** *Per $C \in \{R, B\}, u \in C$ e $t' \leq t$, il guadagno $\Gamma(G, u, (e, e'), t')$ sul Bubble Radius di $u$ ottenuto dallo swap della probabilità di transizione di $(e, e') \in D_C$, è definito come*

$$\Gamma(G, u, (e, e'), t') \doteq B_G^{t'}(u) - B_{G_{e,e'}}^{t'}(u) \tag{3.2}$$

*dove $G_{e,e'}$ è il grafo $G$ dopo aver scambiato i pesi tra $e$ ed $e'$.*

Chiaramente, il guadagno totale può essere ottenuto estendendo la definizione 3.2 all'intero *diversifying swaps.*

**Definizione 11** (Guadagno totale)**.** *Sia $\Sigma \subseteq D_C$ e $G_\Sigma$ il grafo ottenuto scambiando tra loro tutti i pesi delle coppie contenute in $\Sigma$. Il guadagno totale di $\Sigma$ è definito*

$$\mathbb{G}(G, \Sigma) \doteq \frac{1}{|C|} \sum_{u \in C} \left( B_G^t(u) - B_{G_\Sigma}^t(u) \right) \tag{3.3}$$

### 3.2.1 Approssimazione del problema

Risulta evidente che scambiare tra loro tutti gli archi contenuti in $D_C$ diventa un operazione onerosa, costosa e non ragionevole nella pratica. Per questo, si assume che il numero di possibili swaps sia limitato ad un numero $k$.

**Problema 3.** *Dato un grafo $G$, un colore $C$, e un parametro $k$, trovare un set $\Sigma \subseteq D_C$ di dimensione $k$ che massimizzi il guadagno $\mathbb{G}(G, \Sigma)$.*

ShuffLik è l'algoritmo che approssima la soluzione del problema 3.

**Teorema 2** (Approssimazione di ShuffLik)**.** *Sia $\Sigma \subseteq D_C$ l'output di ShuffLik e $OPT$ la soluzione ottima al problema 3. Allora*

$$\mathbb{G}(G, OPT) \leq \left( 2\frac{t}{r}\gamma_{t-2} + 1 \right) \left( 1 + \frac{1}{e} \right) \mathbb{G}(G, \Sigma) \tag{3.4}$$

*dove* $\gamma_t \doteq \max_{v \in V} F_{t'}(v)$ *(2.3)*

Dunque, ShuffLik fornisce una *approssimazione a fattore costante*, sotto l'assunzione che $\gamma_t$ sia una costante e che $r \geq t/2$

## 3.3 ShuffLik: definizione intuitiva

Dato il fatto che la funzione obiettivo (guadagno $\mathbb{G}$) è *monotona* e *sub-modulare*, la scelta della coppia di archi da scambiare può essere effetuata in maniera greedy. La scelta greedy deve considerare la coppia $(e, e')$ che massimizza il guadagno. Ma questa operazione non è per niente banale.
Verrà selezionata, quindi, la coppia $(e, e')$ che *approssima* la scelta greedy. Per far ciò, ShuffLik prediligerà la coppia $(e, e') \in D_C$, con vertice iniziale $v$, che massimizza la quantità $r^{t-2}(v, C) \times |M(e) - M(e')|$, dove $r^{t-2}(v, C)$ è il RWCC di $v$.

## 3.4 ShuffLik: pseudocodice

**Algorithm 2** ShuffLik

**Require:** Grafo $G = (V, E)$, matrice di transizione $M_G$, colore $C \in \{R, B\}$, numero di swaps desiderati $k$.
**Ensure:** L'insieme $\Sigma$ di $k$ coppie di archi che devono essere scambiati.

$\Sigma \leftarrow \emptyset$
**for** $i = 1, \ldots, k$ **do**
  $D_C \leftarrow getDiversifyingSwaps(G, C, \Sigma)$
  $R \leftarrow computeRWCentrality(C)$
  $(e, e') \leftarrow \arg\max_{(e,e') \in D_C} R(v) \times (M_G(e) - M_G(e'))$
  $\Sigma \leftarrow \Sigma \cup \{(e_i, e'_i)\}$
**end for**
**return** $\Sigma$

L'algortimo ShuffLik prende come input un grafo $G$, la matrice di transizione $M_G$, il numero $k$ di swap desiderati e un insieme di nodi di colore $C$.
Per prima cosa, viene creato l'insieme $\Sigma$ (inizialmente vuoto) che conterrà le coppie di archi da scambiare. Successivamente, per $k$ volte: si ottiene l'insieme diversifying swaps $D_C$, utilizzando la funzione *getDiversifyingSwaps*. Quest'ultima prende come parametri il grafo $G$, il colore $C$ e $\Sigma$ e ritorna il set $D_C$ dei possibili swap del grafo $G_\Sigma$, ottenuto scambiando i pesi degli archi in $\Sigma$. Per determinare l'insieme $D_C$, la subroutine dovrà valutare Bubble Radius di tutti i vertici in $C$. In seguito, si calcola il RWCC di ogni nodo in $C$, il quale viene salvato in un dizionario $R$. A questo punto, ShuffLik selezione la coppia $(e, e')$ associata al valore più alto di $R(v) \times |M_G(e) - M_G(e')|$, con $v$ il vertice sorgente di $(e, e')$. Così facendo, si ottiene la coppia di archi $(e, e')$ da aggiungere all'insieme $\Sigma$. Dopo le $k$ iterazioni, infatti, l'insieme $\Sigma$ sarà popolato da $k$ coppie di archi da scambiare.

## 3.5 ShuffLik+

L'algoritmo ShuffLik presuppone che lo scambio dei pesi abbia un costo fisso. Più generalmente, nel caso in cui il costo per effettuare lo swap tra i pesi di due archi non sia un valore fisso e ci sia un budget di costo da rispettare, si utilizza un approccio simile a ShuffLik, ma con un funzione di costo $Q : E \times E \to \mathbb{R}$ e un budget $B \in \mathbb{R}^+$ da non superare. Questa versione modificata dell'algoritmo ShuffLik prende il nome di ShuffLik+.

# Capitolo 4

# Valutazione Sperimentale

In questa capitolo verranno valutate le performance dell'algoritmo RePBubLik+, confrontadolo con altri due algoritmi noti per la riduzione della polarizzazione dei grafi: ROV e Node2Vec.

## 4.1 Algoritmi confrontati

### 4.1.1 ROV(Recommend Opposing View)

L'algoritmo ROV[5] fornisce come output un insieme di $k$ archi da aggiungere a $G$ per minimizzare il controversy score (Random Walk Controversy). Questa è una metrica che caratterizza quanto un topic è "controverso", cioè quanto è ben separato rispetto agli altri topic. L'algoritmo ROV prende come archi candidati quelli che collegano i vertici ad alto grado (con molti archi incidenti) di ciascun "colore" (topic). Gli archi vengono ordinati in ordine decrescente in funzione del loro impatto sul RWC nel grafo, e di questi si individuano i migliori $k$ da aggiungere al grafo G.

### 4.1.2 Node2Vec

L'algoritmo Node2Vec[6] permette di rappresentare un grafo in uno spazio vettoriale a bassa dimensionalità, mantenendo inalterate alcune caratteristiche importanti della rete come, ad esempio, la similiarità tra nodi. La generazione di questo spazio vettoriale si basa sui random walk del grafo. Sullo spazio generato, vengono generalmente eseguiti e addestrati algoritmi di raccomandazione di link. Questo processo permette di migliorare il suggerimento di nuovi collegamenti all'interno del grafo.
Nell' esperimento svolto, per ogni grafo, viene generato uno spazio a 128 dimensioni e, su quest'ultimo, verrà addrestato un algoritmo di regressione logistica. Successivamente, verrà predetta la probabilità di esistenza di ciascun arco uscente da $P(G)$. Al termine, verranno scelti i migliori $k$ archi da aggiungere al grafo $G$, in funzione della loro probabilità.

## 4.2 Dataset

Nelle analisi, vengono generati grafi dai dataset di Amazon e PolBlogs.

- Il dataset di Amazon contiene informazioni sui libri venduti. Ogni libro è un vertice e appartiene ad una categoria (ad un "colore"). Esiste un arco diretto $(u, v)$ nel grafo se $v$ appartiene alla lista di libri simili a $u$ e il suo peso è determinato dal "sales rank" di $v$. Da questo dataset, si ottengono tre grafi formati dalle seguenti coppie di categorie: *Mathematics - Technology*(MaTe), *History of Technology - Military Science*(MiHi) e *Mathematics - Astronomy* (MaAs).

- Il dataset di PolBlogs è una rete diretta di link tra weblogs sulla politica americana. Ogni nodo rappresenta un blog ed è "colorato" in funzione del suo orientamento politico. I link tra i blogs (gli archi nel nostro grafo) sono estratti automaticamente da una scansione della pagina principale dei blog. Il peso di un arco $(v, u)$ è dato dal numero di archi uscenti dal vertice $v$.

Di seguito una tabella riassuntiva delle statistiche dei grafi generati.

| *Amazon* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Topic | $\|V\|$ | $\|R\|$ | $\|B\|$ | $\|E\|$ | $\|E\|_{R\to B}$ | $\|E\|_{B\to R}$ | $\%P_R(G)$ | $\%P_B(G)$ |
| *MaTe* | 1393 | 827 | 566 | 675 | 25 | 42 | 90.91 | 79.63 |
| *MiHi* | 851 | 446 | 405 | 482 | 66 | 63 | 58.33 | 63.46 |
| *MaAs* | 1121 | 827 | 294 | 680 | 11 | 6 | 97.31 | 95.15 |
| *PolBlogs* | | | | | | | | |
| Topic | $\|V\|$ | $\|R\|$ | $\|B\|$ | $\|E\|$ | $\|E\|_{R\to B}$ | $\|E\|_{B\to R}$ | $\%P_R(G)$ | $\%P_B(G)$ |
| *PolBlogs* | 1033 | 545 | 488 | 17348 | 902 | 781 | 87.71 | 90.37 |

**Tabella 4.1:** Tabella delle statisiche dei grafi. La colonna $|V|$ specifica il numero di vertici appartenenti al grafo corrispondente. Le colonne $|R|$ e $|B|$ indicano, rispettivamente, il numero di vertici di colore $R$ e $B$. La colonna $|E|$ indica il numero di archi del grafo corrispondente. Le colonne $|E|_{B\to R}$ e $|E|_{R\to B}$ specificano, rispettivamente, il numero di archi che vanno da nodi di colore $R$ a $B$ e il numero di archi che vanno da nodi di colore $B$ a $R$. Le colonne $\%P_R(G)$ e $\%P_R(G)$ specificano la percentuale di nodi parochial del grafo $G$ rispettivamente di colore $R$ e $B$.

## 4.3 Parametri dei test

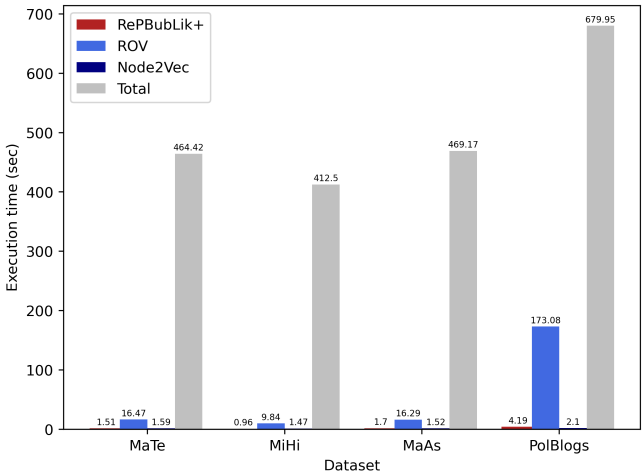
Per ogni grafo, sono stati eseguiti gli algoritmi RePBubLik+, ROV e Node2Vec, con valori crescenti di $K$ (numero di archi da aggiungere) e con valori di $t$, ovvero la lunghezza massima dei random walk. Il valore massimo di $K$ è stato assegnato a 400, per tutti e quattro i grafi in esame. Da notare che, per come è stato ideato lo script degli algoritmi, il numero finale di archi aggiunti può essere ben inferiore al numero $K$ impostato manualmente. Per ogni grafo, infatti, vengono allocati $k_B$ archi per il colore $B$ e $k_R$ archi per il colore $R$, in proporzione alla somma dei Bubble Radius dei vertici parochial di ciascun

colore. Questo metrica permette di aggiungere più archi per il colore avente più vertici parochial. Nei grafi in esame, verranno assegnati circa 200 archi per ogni colore.
Per ciascuna esecuzione, verranno analizzati: i tempi d'esecuzione degli algoritmi, il guadagno $\Delta(G, \Sigma)(2.2)$ e il bias strutturale $\rho(G)(2.1)$.

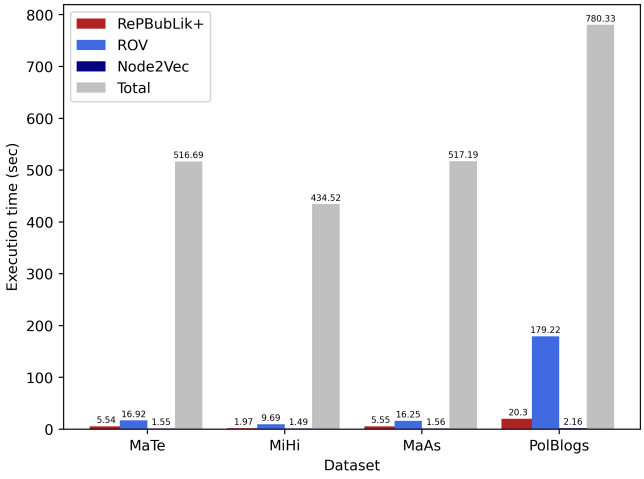
## 4.4 Risultati

### 4.4.1 Tempi di esecuzione

Per valutare l'efficacia in termini di tempo degli algoritmi proposti, sono stati eseguiti 10 test per ciascun grafo e per ciacun valore di $t$. Durante i test, son stati misurati i tempi impiegati dalle varie procedure per determinare i possibili archi candidati. Di questi 10 valori, è stata calcolata la media, ottenendo così una stima qualitativa. Di seguito i grafici riassuntivi dei tempi d'esecuzione.[1]

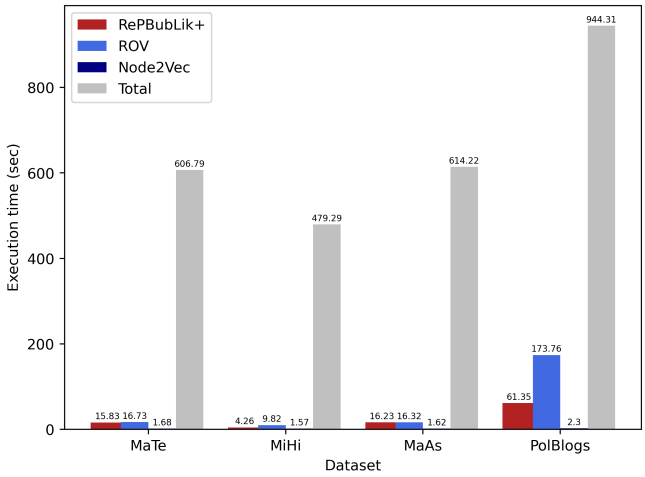



**(a)** Parametro $t = 5$




**(b)** Parametro $t = 10$

[1]Per come è stato ideato lo scpript dagli autori, il tempo totale d'esecuzione comprende alcune varianti di RePBubLik+ che non sono oggetto delle analisi effetuate. Considerando l'esecuzione di queste varianti come un fattore costante, il tempo totale fornisce una stima approssimativa del tempo totale d'esecuzione.




**(c)** Parametro $t = 15$

**Figura 4.1:** Grafici dei tempi d'esecuzione

Analizzando i grafici dei tempi di esecuzione, si possono trarre le seguenti conclusioni:

1. Il tempo impiegato da ciascuna procedura aumenta in relazione alle dimensioni del grafo su cui è applicata.

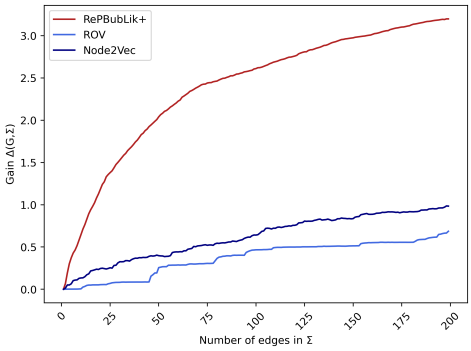
2. Node2Vec risulta essere il migliore in termini di tempo d'esecuzione, per ogni grafo e per ogni valore di $t$.

3. Per ciascun grafo, l'esecuzione di ROV impiega all'incirca lo stesso tempo, indipendentemente dal valore di $t$. Questo è dato dal fatto che ROV utilizza delle metriche differenti dagli altri algoritmi proposti.

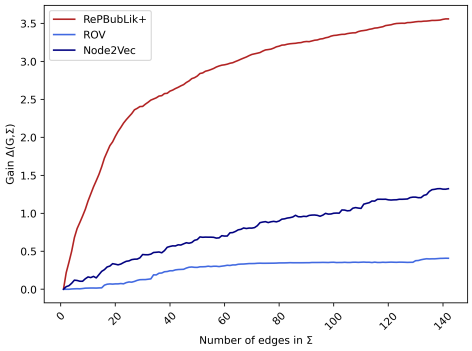
4. RePBubLik+ ha tempistiche intermedie tra quelle di ROV e Node2Vec.

### 4.4.2 Guadagno $\Delta(G, \Sigma)$

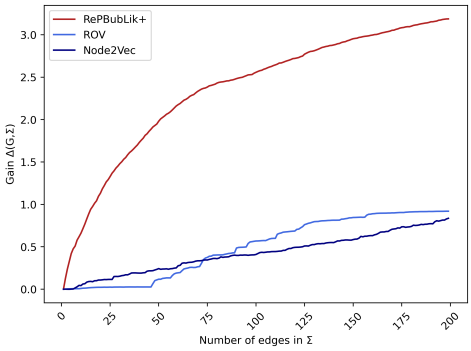
La seconda metrica che viene anlizzata è il il guadagno $\Delta(G, \Sigma)$ (2.2) in relazione al numero degli archi aggiunti al grafo $G$. Lo studio di questa misura è interessante, in quanto permette di valutare il cambiamento medio del Bubble Radius dei vertici parochial $P(G)$.
Nella pagina successiva, i guadagni per ciascun grafo e per ciascun valore di $t$.



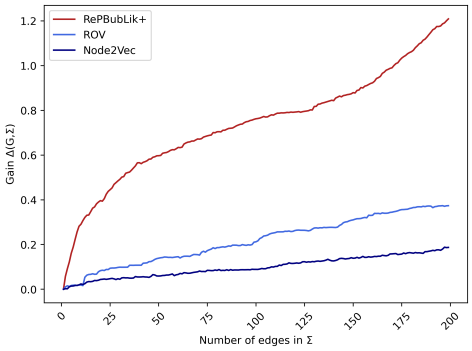

**(a)** *MaTe* plot




**(b)** *MiHi* plot

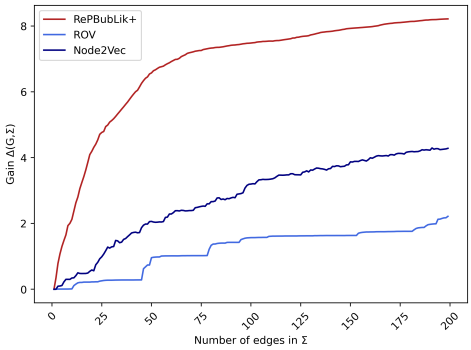



**(c)** *MaAs* plot

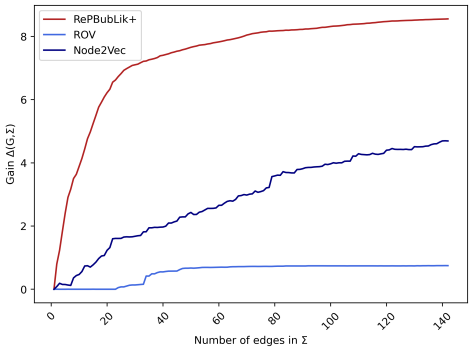



**(d)** *PolBlogs* plot

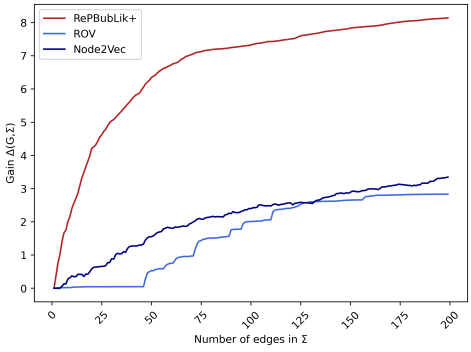
**Figura 4.2:** Grafici $\Delta(G, \Sigma)$ per $t = 5$



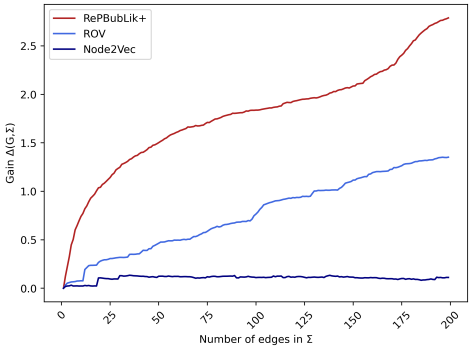

**(a)** *MaTe* plot




**(b)** *MiHi* plot

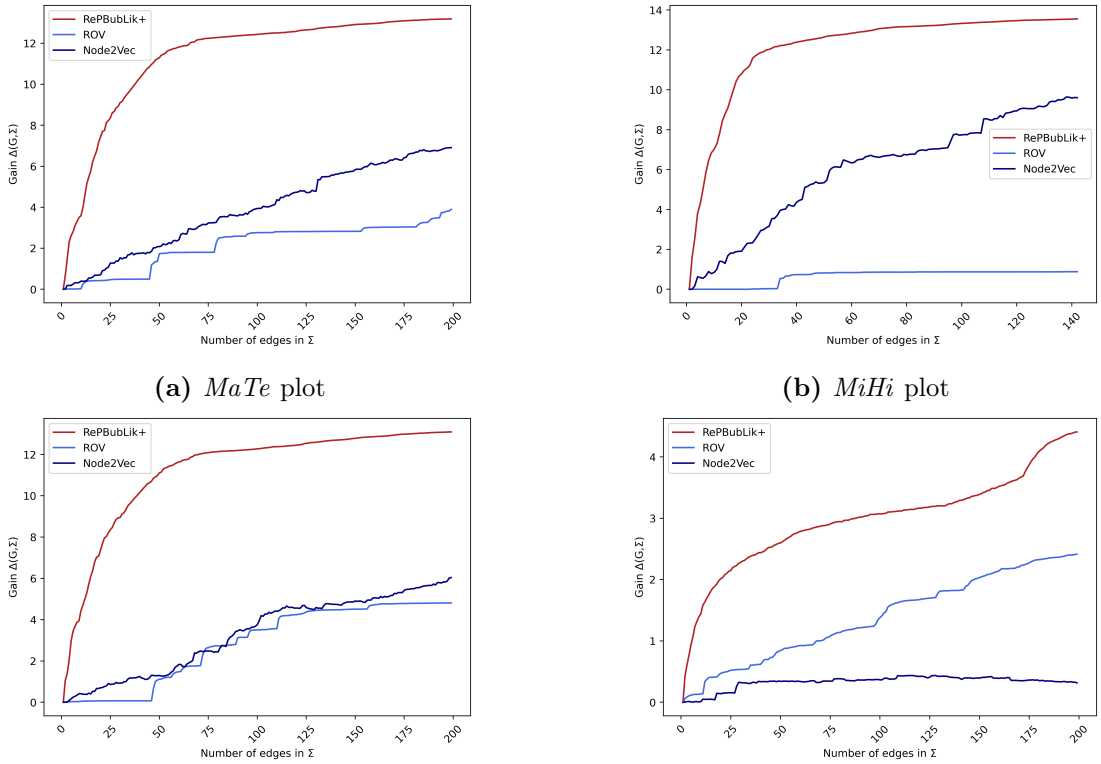



**(c)** *MaAs* plot




**(d)** *PolBlogs* plot

**Figura 4.3:** Grafici $\Delta(G, \Sigma)$ per $t = 10$




**(a)** *MaTe* plot

**(b)** *MiHi* plot

**(c)** *MaAs* plot

**(d)** *PolBlogs* plot

**Figura 4.4:** Grafici $\Delta(G, \Sigma)$ per $t = 15$

Dai grafici dei guadagni, si evince che:

1. RePBubLik+ è il migliore tra gli algoritmi proposti, in termini di guadagno $\Delta(G, \Sigma)$ rispetto al numero di archi in $\Sigma$.

2. L'applicazione di RePBubLik+ a grafi di piccole dimensioni (*MaTe*, *MaAs*, *MiHi*) porta ad un guadagno massimo maggiore rispetto a quello ottenuto mediante gli algoritmi ROV e Node2Vec (es. (*MaAs*) $\frac{\Delta_{R+}}{\Delta_{ROV}} = 2.93$, $\frac{\Delta_{R+}}{\Delta_{Node2Vec}} = 2.56$ per $k = 200, t = 10$).

3. L'applicazione di RePBubLik+ a grafi di medie dimensioni (*PolBlogs*) porta ad un guadagno massimo maggiore rispetto a quello ottenuto mediante gli algoritmi ROV e Node2Vec (es. (*PolBlogs*) $\frac{\Delta_{R+}}{\Delta_{ROV}} = 2.34$, $\frac{\Delta_{R+}}{\Delta_{Node2Vec}} = 18.67$ per $k = 200, t = 10$).

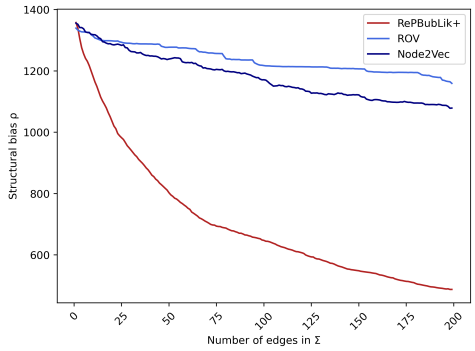
4. RePBubLik+ risulta essere il miglior algoritmo in quanto per un numero ridotto di archi riporta un notevole guadagno: nei grafici sovrastanti, la curva dell'algoritmo RePBubLik+ cresce molto rapidamente per valori piccoli di $k$, raggiungendo una plateau per un certo valore di $\Delta(G, \Sigma)$. Questo, dal punto di vista pratico, è molto interessante perchè permette con pochi archi aggiuntivi di ridurre notevolmente la polarizzazione in grafi simili a quelli presi in esame (es. (*MaTe*) $\frac{\Delta_{R+}}{\Delta_{ROV}} = 7.45$, $\frac{\Delta_{R+}}{\Delta_{Node2Vec}} = 3.35$ per $k = 50, t = 10$).

5. Le considerazioni fatte nei punti precedenti sono generalmente valide, per qualsiasi valore di $t$ e qualsiasi grafo. Valutando i guadagni dei grafi anche in termini di

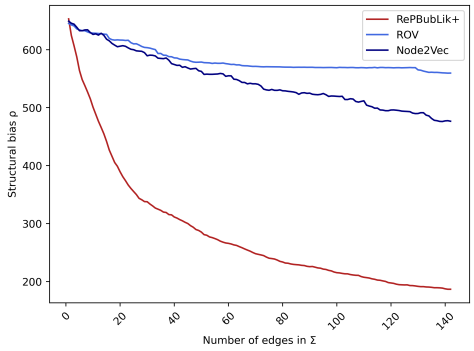
RW massimo, si può notare che per valori di $t$ maggiori, RePBubLik+ raggiunge il plateau con un numero inferiore d'archi (es. (*MaTe*) $\Delta_{R+} = 2$ per $k = 50, t = 5$, $\Delta_{R+} = 6.75$ per $k = 50, t = 10$, $\Delta_{R+} = 11.5$ per $k = 50, t = 15$).

### 4.4.3 Bias strutturale $\rho(G)$

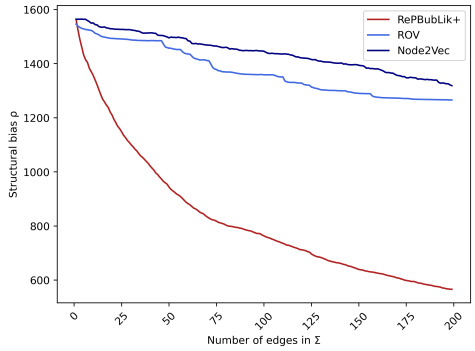
L'ultima metrica che viene anlizzata è il il bias $\rho(G)$ (2.1) in relazione al numero degli archi aggiunti al grafo $G$. Lo studio di questa misura è interessante, in quanto permette di valutare il cambiamento della polarizzazione del grafo $G$.
Di seguito i bias per ciascun grafo e per ciascun valore di $t$.



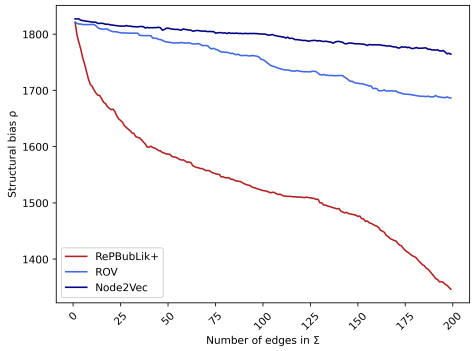

**(a)** *MaTe* plot




**(b)** *MiHi* plot

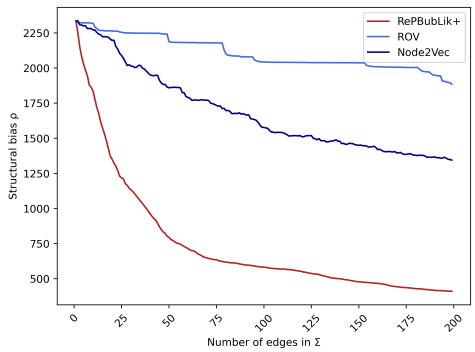



**(c)** *MaAs* plot

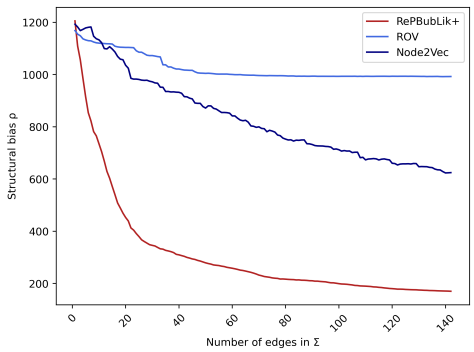



**(d)** *PolBlogs* plot

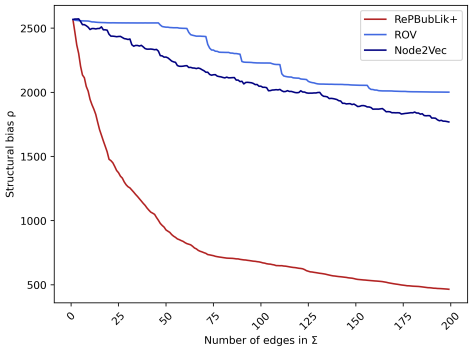
**Figura 4.5:** Grafici $\rho(G)$ per $t = 5$



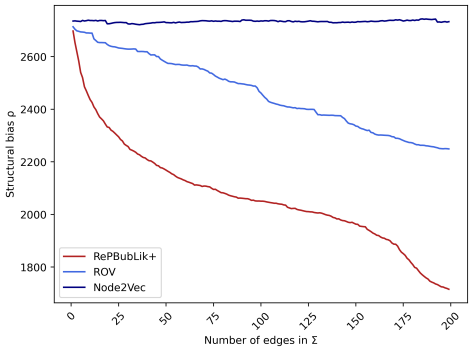

**(a)** *MaTe* plot




**(b)** *MiHi* plot

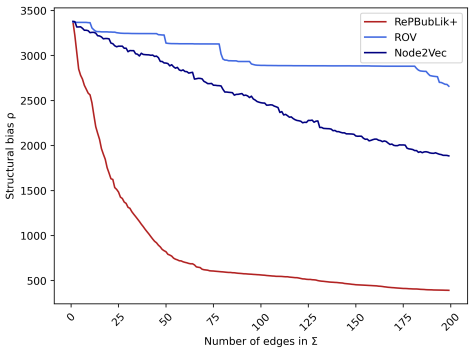



**(c)** *MaAs* plot

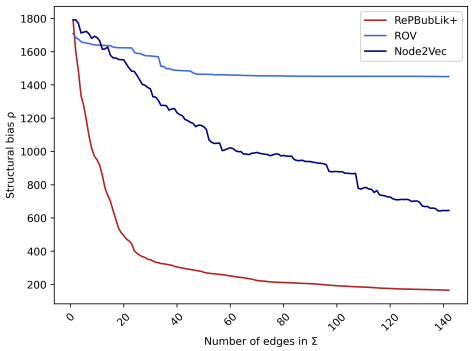



**(d)** *PolBlogs* plot

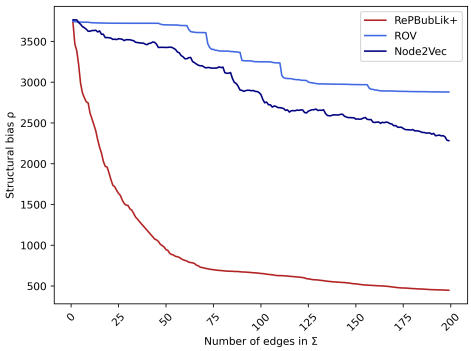
**Figura 4.6:** Grafici $\rho(G)$ per $t = 10$



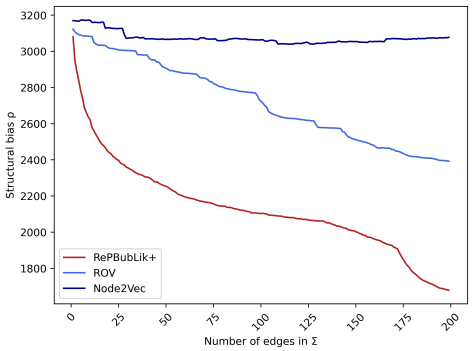

**(a)** *MaTe* plot




**(b)** *MiHi* plot




**(c)** *MaAs* plot




**(d)** *PolBlogs* plot

**Figura 4.7:** Grafici $\rho(G)$ per $t = 15$

Dai grafici dei bias strutturali, si può dedurre che:

1. RePBubLik+ è il migliore tra gli algoritmi proposti, in termini di riduzione di bias $\rho(G)$ rispetto al numero di archi in $\Sigma$.

2. L'applicazione di RePBubLik+ a grafi di piccole dimensioni (*MaTe*, *MaAs*, *MiHi*) porta ad una diminuzione del bias maggiore rispetto a quella ottenuta mediante gli algoritmi ROV e Node2Vec (es. (*MaAs*) $\delta_{\rho_{R+}} = 2700 - 480 = 2220$, $\delta_{\rho_{ROV}} = 2700 - 2000 = 700$, $\delta_{\rho_{Node2Vec}} = 2700 - 1800 = 900$ per $t = 10$).

3. L'applicazione di RePBubLik+ a grafi di medie dimensioni (*PolBlogs*) porta ad una diminuzione del bias maggiore rispetto a quella ottenuta mediante gli algoritmi ROV

e Node2Vec (es. (*PolBlogs*) $\delta_{\rho_{R+}} = 2750 - 1500 = 1250$, $\delta_{\rho_{ROV}} = 2750 - 2250 = 500$, $\delta_{\rho_{Node2Vec}} = 2750 - 2700 = 50$ per $t = 10$).

4. RePBubLik+ risulta essere il miglior algoritmo in quanto per un numero ridotto di archi riporta una notevole diminuzione del bias: nei grafici sovrastanti, la curva dell'algoritmo RePBubLik+ diminuisce molto rapidamente per valori piccoli di $k$, raggiungendo una plateau per un certo valore di $\rho(G)$. Questo, dal punto di vista pratico, è molto interessante perchè permette con pochi archi aggiuntivi di aumentare notevolmente la navigabilità in grafi simili a quelli presi in esame (es. (*MaTe*) $\delta_{\rho_{R+}} = 2300 - 760 = 1540$, $\delta_{\rho_{ROV}} = 2300 - 2200 = 100$, $\delta_{\rho_{Node2Vec}} = 2300 - 1900 = 400$ per $k = 50, t = 10$).

5. Le considerazioni fatte nei punti precedenti sono generalmente valide, per qualsiasi valore di $t$ e qualsiasi grafo. Valutando il bias strutturale dei grafi anche in termini di RW massimo, si può notare che per valori di $t$ maggiori, RePBubLik+ raggiunge il plateau con un numero inferiore d'archi (es. (*MaTe*) $\delta_{\rho_{R+}} = 1350 - 675 = 675$ per $k = 50, t = 5$, $\delta_{\rho_{R+}} = 2300 - 760 = 1540$ per $k = 50, t = 10$, $\delta_{\rho_{R+}} = 3400 - 800 = 2600$ per $k = 50, t = 15$).

### 4.4.4 Conclusioni

Considerando tutti i dati ottenuti dagli esperimenti svolti, RePBubLik+ risulta essere il miglior algoritmo in termini di rapporto tra guadagno e tempo impiegato. Le performance di quest'ultimo, infatti, variano notevolmente a seconda dei parametri forniti: per valori di $t$ maggiori, il guadagno, e la conseguente diminuzione del bias strutturale, aumentano significativamente, a discapito dei tempi di esecuzione. Sarebbe utile, quindi, determinare quale lunghezza massima di RW meglio si addice ad un determinato grafo e quanto sia importante la diminuzione della polarizzazione in quella rete. Per valutare al meglio le performance di RePBubLik+, un'analisi su dataset di dimensioni notevolmente maggiori permetterebbe di esaminare quanto il volume della rete influenzi le performance dell'algoritmo.

# Capitolo 5

# Conclusioni

Questa tesi presenta diversi approcci per la riduzione della polarizzazione nella navigazione nei grafi, ovvero gli algoritmi RePBubLik e ShuffLik. RePBubLik permette di calcolare un'approssimazione accurata per il problema della riduzione del *structural bias*. In seguito alle considerazioni teoriche, son stati svolti dei confronti tra RePBubLik+, la versione più veloce e pratica dell' algoritmo RePBubLik, e le attuali tecniche per la riduzione della polarizzaione dei grafi: ROV e Node2Vec. Dalle analisi, è emerso che RePBubLik+ permette di ridurre drasticamente il *structural bias* del grafo con un numero di archi significativamente inferiore, rispetto agli altri approcci utilizzati.
In aggiunta, è stata brevemente descritta un tecnica alternativa a RePBubLik, che permette di aumentare la navigabilità di una rete effettuando *edge swapping*. Questo algoritmo prende il nome di ShuffLik ed esiste una sua versione più pratica, ShuffLik+. Importante notare che ShuffLik risulta efficace in contesti dove non è possibile eseguire *edge inserction* per motivi di costo o di vincoli funzionali, operando quindi interventi meno invasivi.
Possiamo osservare diverse opportunità di sviluppo futuro:

1. Nella pratica, potrebbe essere difficile stimare le probabilità di transizione all'interno del grafo. I pesi da assegnare agli archi potrebbero essere appresi mediante un algoritmo di machine learning sui dataset.

2. La determinazione di valori realistici del parametro $t$ non è semplice. È un problema rilevante individuare il valore ideale di random walk da fornire all'algoritmo, in funzione del trade-off tra guadagno e tempo di esecuzione.

3. Nell'estensione a più "colori" dell'algoritmo, diventa importante determinare quale coppia di "colori" permette, con un numero ridotto ti archi, di ridurre notevolmente il bias strutturale.

# Bibliografia


[1] Shahrzad Haddadan, Cristina Menghini, Matteo Riondato, Eli Upfal (2022) *Reducing polarization and increasing diverse navigability in graphs by inserting edges and swapping edge weights. In Proceedings of ACM WSDM'21.* https://link.springer.com/article/10.1007/s10618-022-00875-8

[2] Cristina Menghini (2021) *Reducing Polarization and Improving Diverse Navigability* https://github.com/CriMenghini/RePBubLik

[3] Amazon (2006) *Amazon product co-purchasing network metadata.* https://snap.stanford.edu/data/amazon-meta.html

[4] PolBlogs (2005) *A directed network of hyperlinks between weblogs on US politics.* http://www-personal.umich.edu/~mejn/netdata/

[5] Kiran Garimella, Gianmarco De Francisci Morales, Aristides Gionis, and Michael Mathioudakis (2017) *Reducing Controversy by Connecting Opposing Views. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM'17).* https://arxiv.org/abs/1611.00172

[6] Aditya Grover, Jure Leskovec (2016) *node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* https://cs.stanford.edu/~jure/pubs/node2vec-kdd16.pdf

[7] Wikipedia (2023) *Graph theory* https://en.wikipedia.org/wiki/Graph_theory

[8] Wikipedia (2022) *Random walk closeness centrality* https://en.wikipedia.org/wiki/Random_walk_closeness_centrality