



UNIVERSITÀ DEGLI STUDI  
DI SALERNO

## SMiLe-CoDe

Progetto "Reti Sociali"

**Luigina Costante**

**Salvatore Michele De Luca**

Dipartimento di Informatica  
Laurea Magistrale in Data Science & Machine Learning

Fisciano, giugno 2025



## UNIVERSITÀ DEGLI STUDI DI SALERNO

### SMiLe-CoDe

Progetto "Reti Sociali"

**Luigina Costante**

**Salvatore Michele De Luca**

Dipartimento di Informatica

Laurea Magistrale in Data Science & Machine Learning

Fisciano, giugno 2025

# 1

## Definizione del problema

L'influenza sociale rappresenta il processo mediante il quale gli individui modificano i propri comportamenti in base alle interazioni che hanno con altre persone, conformandosi con esse. Il presente studio prevede un'analisi basata sul fenomeno *Majority cascade in cost networks*, dove l'obiettivo è quello di trovare un seed set  $S$  che abbia un costo totale inferiore ad una certa soglia  $k$  e che massimizzi l'influenza ottenibile sulla rete. Questo è problema è NP-hard ( $O(2^{\log^{(1-\epsilon)} n})$ ); per renderlo compatibile con i vincoli computazionali sono state sviluppate delle euristiche, che verranno approfondite in seguito. Prima di definire formalmente il problema diamo alcune definizioni.

*Quando le persone sono connesse da una rete, diventa possibile che esse influenzino vicendevolmente i loro comportamenti e le loro decisioni. (Easley et al., 2010).*

Questa frase rappresenta una buona sintetizzazione di ciò che è l'influenza all'interno delle reti sociali, permettendoci di astrarre il concetto alle sue componenti principali. Quando parleremo di persone connesse da una rete (*network*) faremo riferimento alle cosiddette reti sociali (*social networks*), particolari tipologie di reti che hanno lo scopo di mappare relazioni di varia natura tra individui.

In generale, si definisce una rete come una qualsiasi collezione di oggetti che sono legati tra loro tramite dei *link*. La flessibilità di questa definizione ci concede diverse tipologie di reti e possibili astrazioni.

Una rete è solitamente astratta mediante l'utilizzo di grafi. Definiamo un grafo  $G = (V, E)$  consistente di:

- un insieme  $V$  di nodi;
- un insieme  $E$  di archi  $e = (v, u)$  dove  $v, u \in V$ .

Inoltre definiamo per ogni vertice  $v \in V$ :

- $N(v)$  l'insieme dei nodi adiacenti di  $v$  in  $G$ , dove  $u$  e  $v$  sono adiacenti se  $\exists(u, v) \in E$ ;

- $d(v)$  il grado di  $v$  in  $G$ , cioè  $d(v) = |N(v)|$ .

Nel caso dei social network, i nodi rappresentano persone e gli archi relazioni (link) tra di esse. Nelle reti sociali una delle dinamiche più studiate è quella dell'influenza, ovvero la capacità di un individuo o di un insieme di individui di popolarizzare un comportamento nella rete. Tali processi sono associati al concetto di *cascading*, ovvero allo studio della diffusione di informazioni, malattie, comportamenti e simili all'interno di una rete; questo tema è stato approfondito nel Capitolo V di *Networks, Crowds, and Markets: Reasoning about a Highly Connected World* (Easley et al., 2010). È importante evidenziare che durante un processo di cascading, ad ogni step un insieme di nodi influenzati ne può influenzare altri. Per semplificare l'analisi nei passaggi successivi, assumeremo che un nodo, una volta influenzato, rimanga tale in modo permanente, e che dunque l'insieme dei nodi influenzati non possa ridursi nel tempo.

Un altro concetto importante è quello relativo al *Majority domination*. Dato un grafo  $G = (V, E)$ , un *dominating set* (Cabrera-Martínez, 2022)  $D$  di  $G$  è un sottoinsieme di vertici di  $G$  tale che

$$\forall v \in V \setminus D, \quad |N(v) \cap D| \geq 1$$

ovvero ogni nodo non in  $D$  ha almeno un suo adiacente nel dominating set. Partendo da questa definizione, un **majority dominating set** in  $G$  è definito come un sottoinsieme  $D \subseteq V$  tale che per ogni nodo  $v \in V \setminus D$ , il numero di vicini di  $v$  appartenenti a  $D$  sia almeno pari alla metà (arrotondata per eccesso) del grado di  $v$ :

$$\forall v \in V \setminus D, \quad |N(v) \cap D| \geq \left\lceil \frac{d(v)}{2} \right\rceil.$$

Infine, possiamo definire il **Majority Cascade in Cost Networks** nel seguente modo. Dato un network  $G = (V, E)$  e un seed set  $S \subseteq V$ , un processo dinamico di diffusione dell'influenza secondo la maggioranza (Majority Cascade) su  $G$  è descritto come una sequenza di sottoinsiemi:

$$\text{Inf}[S, 0], \text{Inf}[S, 1], \dots, \text{Inf}[S, r], \dots \subseteq V$$

dove:

- $0, 1, r$  rappresentano gli istanti (o step) del processo di cascading;
- $\text{Inf}[S, 0] = S$ ;
- $\text{Inf}[S, r] = \text{Inf}[S, r - 1] \cup \left\{ v \in V \setminus \text{Inf}[S, r - 1] : |N(v) \cap \text{Inf}[S, r - 1]| \geq \left\lceil \frac{d(v)}{2} \right\rceil \right\}$ .

Il cascade termina all'istante  $t$ , definito come il più piccolo istante tale che:

$$\text{Inf}[S, t] = \text{Inf}[S, t + 1].$$

Nel caso specifico delle *cost networks*, che costituiscono il contesto del problema qui analizzato, al grafo  $G$  è associata una funzione di costo  $c : V \rightarrow \mathbb{N}$ . Questa funzione

assegna a ciascun nodo  $v \in V$  un costo, permettendo così di valutare il costo totale di un seed set  $S$  come segue:

$$c(S) = \sum_{u \in S} c(u).$$

Dato un budget  $k$ , l'obiettivo è determinare un seed set  $S$  tale che  $c(S) \leq k$ , massimizzando la cardinalità dell'insieme  $\text{Inf}[S]$ , ovvero il numero totale di nodi influenzati al termine del processo.

Come già discussso, questo è un problema *NP-hard*. Per studiarne il comportamento, si è scelto di analizzare l'influenza ottenibile al variare del budget, selezionando per ciascun valore di  $k$  un seed set massimale che non ne superi il limite. Più formalmente:

*dato un budget  $k$ , l'obiettivo è determinare un seed set massimale  $S$  tale che  $c(S) \leq k$ , e valutare  $|\text{Inf}[S]|$  al variare di  $k$ .*

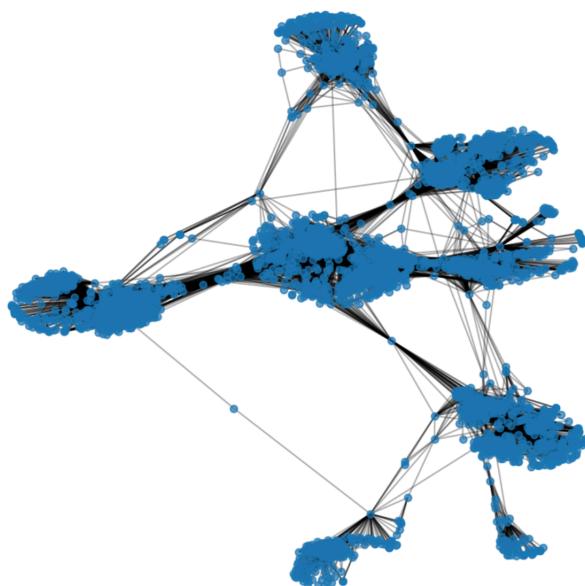
# 2

## Descrizione della rete scelta

La rete *ego-Facebook* è stata reperita dal database SNAP ([Leskovec et al., 2014](#)), nella sezione dedicata ai *Social Circles*. Questo set di dati è costituito da "cerchie" (o "liste di amici") di Facebook. Il dataset originale include diverse informazioni: caratteristiche dei nodi (profili), cerchie e reti ego-centriche.

Per il presente lavoro, è stata utilizzata unicamente la lista degli archi, necessaria per la costruzione del grafo, in quanto le informazioni aggiuntive risultavano superflue rispetto agli obiettivi dell'analisi.

La Figura 2.1 mostra il grafo della rete sociale oggetto di analisi. I nodi sono disposti secondo una configurazione spaziale che evidenzia chiaramente la presenza di sottostruuture coerenti con la formazione di comunità. I nodi densamente connessi tendono a raggrupparsi visivamente, mettendo in risalto l'elevato coefficiente di clustering osservato, il quale verrà approfondito in seguito.



**Figura 2.1:** Grafo completo rete sociale ego-Facebook

## 2.1 Analisi statistica della rete

Le statistiche di rete che hanno guidato la scelta di questa struttura sono riportate nella Tabella 2.1. Nello specifico, la rete è composta da 4 039 nodi e 88 234 archi non orientati, con un coefficiente di clustering pari a 0,6055. Queste caratteristiche rendono la rete moderatamente densa e caratterizzata dalla presenza di comunità, ossia sottogruppi di nodi fortemente connessi. Questa struttura implica che processi come la diffusione di informazioni o di epidemie tendano ad essere più rapidi all'interno dei singoli cluster, ma possano procedere più lentamente attraverso cluster differenti.

Statistica	Valore
Nodi totali	4 039
Archi totali	88 234
Nodi nel WCC più grande	4 039 (1,000)
Archi nel WCC più grande	88 234 (1,000)
Nodi nel SCC più grande	4 039 (1,000)
Archi nel SCC più grande	88 234 (1,000)
Coefficiente di clustering medio	0,6055
Numero di triangoli	1 612 010
Frazione di triangoli chiusi	0,2647
Diametro (lunghezza max del cammino minimo)	8
Diametro effettivo (90° percentile)	4,7

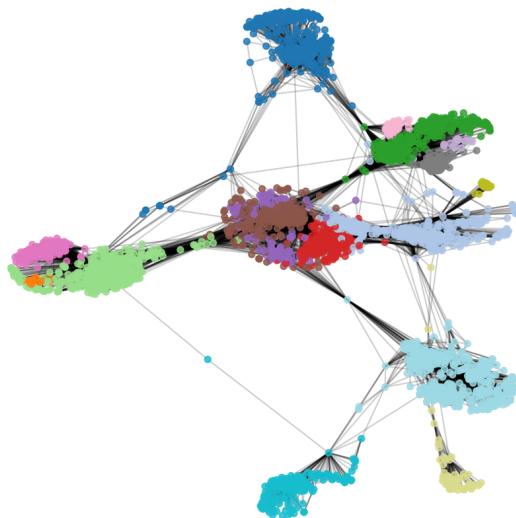
Tabella 2.1: Statistiche principali del set di dati

Inoltre, sia la *Weakly Connected Component* (WCC) che la *Strongly Connected Component* (SCC) comprendono la totalità dei nodi della rete, a dimostrazione del fatto che il grafo risulta connesso. Tale proprietà è particolarmente desiderabile, poiché consente di aspirare a una *complete cascade* senza la necessità di introdurre euristiche specifiche per la gestione di componenti disconnesse.

La scelta di questa rete è stata inoltre motivata da due ulteriori considerazioni: in primo luogo, la rete presenta dimensioni tali da consentire analisi computazionalmente sostenibili, un aspetto rilevante in presenza di risorse computazionali limitate. In secondo luogo, la rete presenta proprietà topologiche tipiche delle reti di relazioni umane (elevato clustering, comunità ben definite), un contesto conforme ai modelli di diffusione di informazioni o comportamenti considerati in questo studio, come il processo di *majority cascade*.

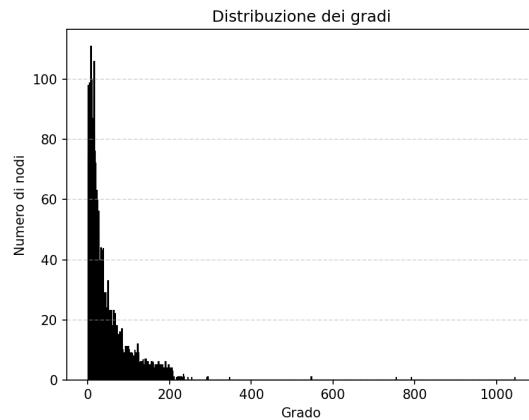
## 2.2 Analisi grafica della rete

A complemento delle statistiche riportate nella Tabella 2.1, si presentano di seguito alcune visualizzazioni grafiche che permettono di approfondire le caratteristiche strutturali della rete.



**Figura 2.2:** Partizionamento della rete in comunità.

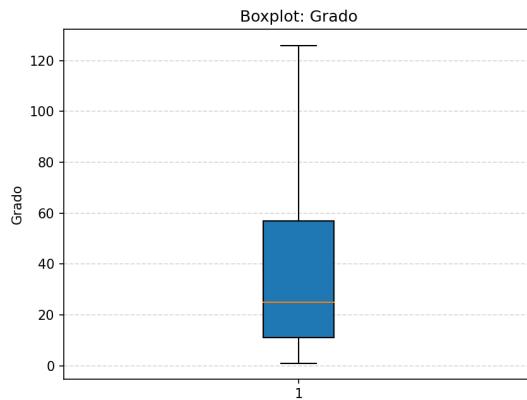
La Figura 2.2 evidenzia la segmentazione della rete in comunità, ottenuta tramite l'algoritmo Louvain (Blondel et al., 2008). Ciascun colore rappresenta una comunità distinta, e la mappa mostra che la rete è fortemente modulare, ovvero suddivisa in sottogruppi coesi, in linea con la natura delle relazioni sociali.



**Figura 2.3:** Distribuzione dei gradi dei nodi.

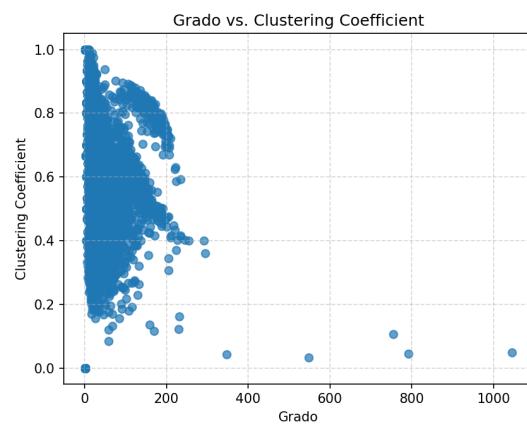
L'istogramma in Figura 2.3 mostra la distribuzione dei gradi dei nodi. La rete evidenzia una distribuzione fortemente asimmetrica e decrescente, tipica delle reti sociali reali, dove pochi nodi (**hub**) sono caratterizzati da numerose connessioni mentre la maggior parte di essi presentano un grado più basso.

Il boxplot relativo al grado dei nodi in Figura 2.4, fornisce un riassunto statistico (mediana, quartili, outlier) che conferma l'alta dispersione e la presenza di nodi con grado significativamente superiore alla media.



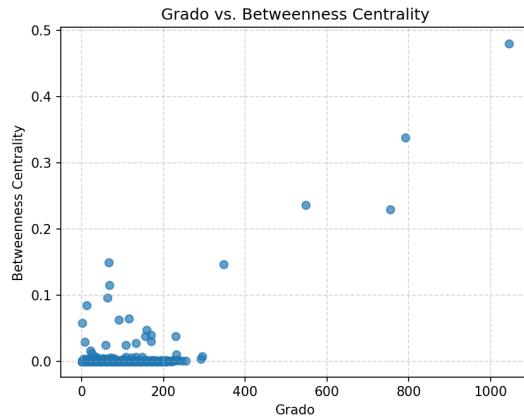
**Figura 2.4:** Boxplot dei gradi della rete.

Il grafico in Figura 2.5 mostra la relazione tra il grado di ciascun nodo e il relativo coefficiente di clustering. Si osserva una tendenza inversa: i nodi con grado elevato tendono ad avere un clustering più basso, indicando che gli hub fungono da ponte tra comunità distinte.



**Figura 2.5:** Relazione tra grado e coefficiente di clustering.

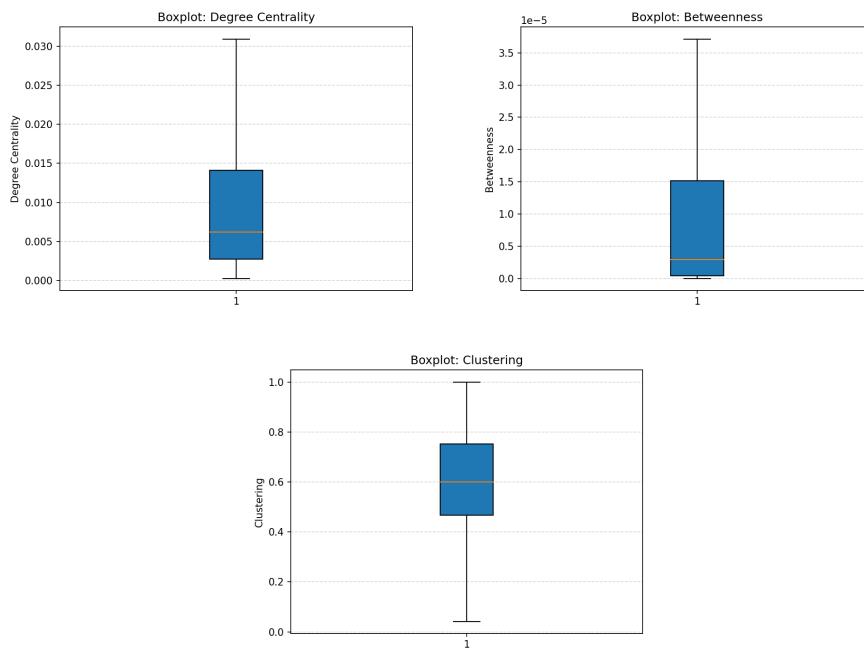
Lo scatter plot riportato in Figura 2.6 confronta il grado con la betweenness centralità. Come atteso, si nota una correlazione positiva, sebbene non perfetta: i nodi con alto grado spesso presentano anche una maggiore centralità intermedia, ma esistono eccezioni dovute alla topologia locale.



**Figura 2.6:** Confronto tra grado e betweenness centrality.

La Figura 2.7 rappresenta i boxplot delle principali metriche di centralità e coesione della rete: centralità di grado, betweenness centrality e coefficiente di clustering. Le distribuzioni risultano marcatamente asimmetriche, con valori medi relativamente bassi ma code estese verso l’alto, a indicare la presenza di pochi nodi con valori significativamente superiori rispetto alla mediana.

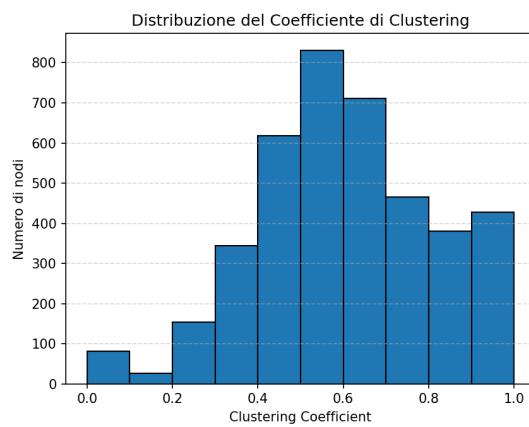
Tale configurazione suggerisce che una minoranza di nodi occupa posizioni strutturalmente rilevanti, sia in termini di connessioni dirette (alto grado), sia come elementi di snodo nel passaggio di informazioni (alta betweenness), sia come nuclei di elevata coesione locale (alto clustering). Al contrario, la maggior parte dei nodi si colloca in posizioni periferiche, con bassa intermediazione e scarsa influenza topologica.



**Figura 2.7:** Boxplot relativi principali metriche: degree centrality, betweenness centrality e coefficiente di clustering.

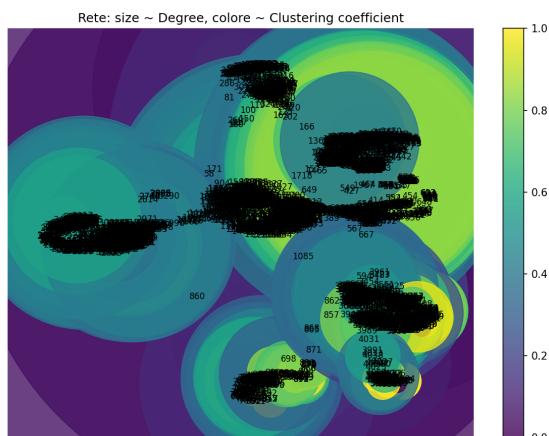
Questo comportamento riflette fedelmente la natura ego-centrica della rete, nella quale l'intero grafo è costruito attorno a un nodo principale (l'**ego**) e alle sue connessioni dirette. La presenza di nodi centrali molto influenti (tipicamente amici diretti dell'ego con molti legami reciproci) affiancati da una molitudine di nodi con ruoli marginali è, dunque, un tratto strutturale atteso e coerente con il paradigma delle reti sociali personali.

L'istogramma dei coefficienti di clustering presente in Figura 2.8 mostra che una buona parte dei nodi presenta valori elevati, confermando una marcata tendenza alla formazione di triadi chiuse. Questo dato è coerente con la natura della rete sociale in analisi, dove la probabilità che due amici di un nodo siano anch'essi amici è alta.



**Figura 2.8:** Distribuzione dei coefficienti di clustering.

Infine, in Figura 2.9 è riportata una visualizzazione della rete che evidenzia i nodi in base al loro coefficiente di clustering. I colori o le dimensioni dei nodi variano in funzione del clustering locale, mettendo in risalto le zone della rete caratterizzate da una maggiore densità di connessioni reciproche.



**Figura 2.9:** Visualizzazione della rete con enfasi sul clustering locale.

## 2.3 Analisi local bridges

In questa fase si è proceduto ad analizzare le informazioni di connettività associate alla rete *ego-Facebook* considerando le due seguenti definizioni.

*Un bridge del grafo  $G$  è un arco  $e = (u, v)$  tale che, la sua rimozione disconnette le componenti di  $u$  e  $v$ . Ovvero, è impossibile raggiungere  $u$  da  $v$  se l'edge  $e$  viene reciso.*

*Un local bridge rappresenta un arco  $e = (u, v) \in E$  i cui estremi  $u$  e  $v$  non hanno amici in comune, ovvero se la rimozione dell'arco aumenterebbe la distanza tra  $u$  e  $v$  a un valore strettamente maggiore di due.*

In primo luogo, sono stati individuati 78 bridge locali mediante la funzione `local_bridges()` della libreria **Networkx**. Di questi, 75 sono caratterizzati da uno span infinito - ovvero, la distanza minima tra i due nodi connessi da quell'arco diverge (diventa infinita) se l'arco viene rimosso - per cui risultano essere anche veri e propri bridge della rete.

Poiché sappiamo che il grafo è connesso, possiamo concludere che, per ciascuno di questi bridge, almeno uno dei due nodi è collegato al resto della rete. Tuttavia, il fatto che lo span sia infinito indica che l'altro nodo, senza quell'arco, non è più raggiungibile: ciò implica che è isolato dal resto del grafo e dipende interamente da quel collegamento.

Questi bridge rappresentano quindi un collegamento tra un nodo periferico (o isolato) e un nodo che funge da punto di accesso alla rete. Di conseguenza, esercitare un'influenza sul nodo "ponte" permette indirettamente di influenzare anche il nodo isolato collegato ad esso.

### 2.3.1 Statistiche della rete senza bridge

Per analizzare l'effettivo contributo dei local bridges nella rete in esame, è stato definito un nuovo insieme di archi (*edge list*) escludendo quelli classificati come local bridges. Successivamente, quest'ultimo è stato utilizzato per il calcolo delle statistiche del grafo risultante, riportate nella Tabella 2.2.

Statistica	Valore
Nodi totali	3 963
Archi totali	88 156
Nodi nel WCC più grande	3 963 (1,000)
Archi nel WCC più grande	88 156 (1,000)
Nodi nel SCC più grande	3 963 (1,000)
Archi nel SCC più grande	88 156 (1,000)
Coefficiente di clustering medio	0.6172
Numero di triangoli	1 612 010
Frazione di triangoli chiusi	0,521248
Diametro (lunghezza max del cammino minimo)	8
Diametro effettivo (90° percentile)	4,8

**Tabella 2.2:** Statistiche principali del set di dati con i local bridge rimossi

Il nuovo grafo  $G'$  conta 3 963 nodi, ovvero 76 in meno rispetto alla rete originale. Questa riduzione è dovuta alla rimozione dei local bridges, che ha isolato 75 nodi: ciascuno di questi era connesso al resto della rete esclusivamente tramite uno di tali archi. Inoltre, è stato perso un nodo addizionale che fungeva da ponte tra un nodo connesso e uno isolato; la rimozione simultanea dei bridge che lo coinvolgevano ha fatto sì che non comparisse più in alcuna tupla dell'insieme  $E$  (insieme degli archi).

Nonostante la rimozione di questi collegamenti, la struttura globale della rete è rimasta intatta. In particolare, le 18 comunità individuate in precedenza risultano ancora tutte raggiungibili: nessuna componente è stata disconnessa. Ciò a dimostrazione del fatto che i bridges fungono da ponte esclusivamente a punti isolati. Questo suggerisce che, pur essendo i collegamenti tra le comunità meno densi rispetto a quelli interni, la rete conserva una connettività sufficiente a evitare dipendenza critica da singoli archi (local bridges) per il mantenimento della coesione complessiva.

# 3

## Metodologia

In questo capitolo sono delineati i passi definiti per l'analisi effettuata nel presente lavoro a partire dalla definizione delle funzioni di costo utilizzate, per poi passare alla descrizione del budget e agli algoritmi implementati.

### 3.1 Funzioni di costo

Sono state implementate tre diverse funzioni di costo  $c(v)$  per ciascun nodo  $v \in V$ , al fine di valutare le performance degli algoritmi proposti in scenari differenti. Ogni funzione è stata utilizzata come parametro di input nei tre algoritmi analizzati. Di seguito si descrivono nel dettaglio:

- **Cost1:**  $c_1(v) = \lceil d(v)/2 \rceil$ , dove  $d(v)$  è il grado del nodo  $v$ . Questa funzione rappresenta una stima naturale del costo basata sulla struttura locale del grafo: più un nodo ha connessioni, più risulta "costoso" da attivare.
- **Cost2:**  $c_2(v) \in \mathbb{Z}$ , valore intero casuale generato uniformemente nell'intervallo  $[\min(c_1), \max(c_1)]$ . L'obiettivo di questa funzione è analizzare il comportamento degli algoritmi in presenza di costi non correlati alla struttura del grafo, ma comunque all'interno di un range plausibile, ovvero definito a partire dai valori estremi della funzione deterministica  $c_1$ .
- **Cost3:**

$$c_3(v) = \left( \log(\text{bt}(v) + \varepsilon) - \min_{u \in V} \log(\text{bt}(u) + \varepsilon) \right) \cdot \frac{\max c_1}{\max_{u \in V} (\log(\text{bt}(u) + \varepsilon) - \min_{z \in V} \log(\text{bt}(z) + \varepsilon))}$$

dove  $\text{bt}(v)$  rappresenta la betweenness centrality del nodo  $v$  e  $\varepsilon$  è una costante positiva molto piccola, aggiunta per evitare logaritmi nulli. Il valore del logaritmo è traslato per avere minimo 0 e poi scalato per avere lo stesso range della funzione  $c_1$ , rendendo così confrontabili le tre funzioni. Questa funzione assegna costi maggiori ai nodi più centrali nella rete, simulando scenari in cui tali nodi sono più complessi (e quindi più costosi) da attivare.

Questa funzione di costo si basa sulla metrica della *betweenness centrality* (Zhang et al., 2017), scelta per identificare i nodi chiave nel successivo processo di diffusione. Tale metrica fornisce una misura della centralità di un nodo in termini di frequenza con cui esso compare nei cammini minimi tra tutte le coppie di nodi del grafo, indicando quindi la sua importanza strategica all'interno della rete.

La prima funzione può essere considerata una sorta di "baseline" per l'analisi poiché è direttamente legata alla topologia del grafo, deterministica e permette un confronto chiaro tra le strategie algoritmiche. Le altre due funzioni, invece, introducono varianza (casualità e centralità) per testare la robustezza degli algoritmi in contesti più realistici e non uniformi.

## 3.2 Dettagli sulla scelta del budget

Per ciascuna funzione di costo  $c(v)$ , è stato definito un intervallo specifico di valori di budget entro cui testare gli algoritmi. Tale intervallo non è stato scelto in modo arbitrario, ma è stato calcolato in modo adattivo rispetto alla distribuzione dei costi nel grafo.

In particolare, il range di variazione del budget è determinato come segue:

- **Budget minimo:**

$$\text{min\_budget} = \max_{v \in V} c(v)$$

Ovvero, il valore minimo del budget corrisponde al costo massimo tra tutti i nodi del grafo. Questo assicura che almeno un nodo sia selezionabile come seed.

- **Budget massimo:**

$$\text{max\_budget} = \sum_{v \in V} c(v)$$

Il valore massimo del budget rappresenta il caso teorico in cui tutti i nodi possano essere selezionati, cioè quando il budget è sufficiente a coprire il costo dell'intero grafo.

Questa scelta rende il range del budget coerente e significativo per ogni funzione di costo adottata, poiché si adatta automaticamente alla distribuzione dei costi generati. In questo modo, i confronti tra le diverse strategie algoritmiche risultano più equi, in quanto ogni configurazione viene valutata all'interno di un range proporzionato al costo effettivo dei nodi. Inoltre, tale approccio permette di analizzare sia scenari vincolati (bassi budget) che scenari in cui "l'influencer" può operare con ampia disponibilità di risorse (alti budget).

## 3.3 Cost Seeds Greedy (CSG)

L'Algoritmo 1 Cost-Seeds-Greedy, definito *Pervasive Partial Domination (PPD)* in Cordasco et al., 2022, è una strategia euristica greedy pensata per selezionare un sottoinsieme

di nodi (seed set) in un grafo  $G = (V, E)$ , massimizzando una funzione di diffusione  $f_i$  sotto vincoli di budget. Ogni nodo  $v \in V$  è associato a un costo  $c(v)$ , e l'obiettivo è costruire un insieme di seed  $S_p$  tale che il costo totale  $c(S_p)$  non superi un budget  $k$ , e la diffusione dell'informazione, rappresentata da  $f_i(S_p)$ , sia massimizzata. L'algoritmo parte da un insieme vuoto e iterativamente aggiunge il nodo  $u$  che massimizza il rapporto tra il guadagno marginale di diffusione  $\Delta_u f_i(S_d)$  e il costo  $c(u)$ , fino a superare il budget. Al termine, viene restituito l'insieme di nodi ottenuto prima dell'ultima aggiunta che ha violato il vincolo di costo. L'efficacia dell'approccio dipende dalla scelta della funzione  $f_i$ , che può modellare differenti scenari di diffusione come il numero di nodi influenzati o altre metriche di copertura.

---

**Algorithm 1** Cost-Seeds-Greedy( $G, k, c, f_i$ )

---

**Input:** A graph  $G = (V, E)$ , a budget  $k$ , a cost function  $c(v)$ , an influence function  $f_i$

**Output:** A seed set  $S_p$

- 1:  $S_p = S_d = \emptyset$
  - 2: **repeat**
  - 3:     Select  $u = \arg \max_{v \in V \setminus S_d} \left\{ \frac{\Delta_v f_i(S_d)}{c(v)} \right\}$
  - 4:      $S_p = S_d$
  - 5:      $S_d = S_p \cup \{u\}$
  - 6: **until**  $c(S_d) > k$
  - 7: **return**  $S_p$
- 

**Funzioni obiettivo** Nello specifico, sono state considerate tre funzioni obiettivo:  $f_1$ ,  $f_2$  e  $f_3$ . Quest'ultime sono progettate per quantificare l'efficacia di un insieme di nodi  $S \subseteq V$  nel contribuire alla diffusione all'interno del grafo. Ciascuna funzione si basa sul numero di vicini che un nodo ha all'interno di  $S$ , confrontato con una soglia definita in base al grado  $d(v)$  del nodo. Le tre funzioni sono definite come segue:

$$f_1(S) = \sum_{v \in V} \min \left\{ |N(v) \cap S|, \left\lceil \frac{d(v)}{2} \right\rceil \right\} \quad (3.1)$$

$$f_2(S) = \sum_{v \in V} \sum_{i=1}^{|N(v) \cap S|} \max \left\{ \left\lceil \frac{d(v)}{2} \right\rceil - i + 1, 0 \right\} \quad (3.2)$$

$$f_3(S) = \sum_{v \in V} \sum_{i=1}^{|N(v) \cap S|} \max \left\{ \frac{\left\lceil \frac{d(v)}{2} \right\rceil - i + 1}{d(v) - i + 1}, 0 \right\} \quad (3.3)$$

La funzione  $f_1$  si basa su un criterio semplice e locale: per ogni nodo  $v \in V$ , si conteggiano i vicini già presenti in  $S$  fino a un massimo pari a  $\left\lceil \frac{d(v)}{2} \right\rceil$ , ovvero la soglia di attivazione per la maggioranza. La funzione  $f_2$  fornisce una stima più fine, pesando ogni vicino in  $S$  in base alla sua posizione nell'ordine di attivazione. La funzione  $f_3$  in-

troduce un ulteriore livello di dettaglio normalizzando il contributo di ciascun vicino rispetto al grado residuo del nodo.

Sebbene  $f_2$  e  $f_3$  offrano una modellazione più accurata della dinamica di diffusione, in questo lavoro si è scelto di adottare la funzione  $f_1$  per guidare l'algoritmo *Cost-Seeds-Greedy*. La scelta è motivata dalla maggiore semplicità computazionale di  $f_1$ , che consente di ridurre i tempi di calcolo mantenendo comunque una buona capacità discriminativa tra i nodi.

### 3.3.1 Ottimizzazioni apportate

Per migliorare l'efficienza computazionale dell'algoritmo *Cost-Seeds-Greedy*, sono state introdotte diverse ottimizzazioni, mirate a ridurre la complessità computazionale per ogni iterazione del processo greedy.

In particolare, è stata implementata una funzione incrementale, `compute_delta`, la quale calcola la variazione marginale di guadagno  $\Delta$  di un nodo  $v$  rispetto alla funzione submodulare scelta. Piuttosto che ricalcolare interamente il contributo marginale della funzione a ogni passo, la variazione viene determinata confrontando il numero di vicini già attivati prima e dopo l'inclusione ipotetica di  $v$ . Ad esempio, nel caso della funzione submodulare  $f(S) = \sum_{v \in V} \min\{|N(v) \cap S|, \lceil d(v)/2 \rceil\}$ , la variazione marginale si riduce a:

$$\Delta_v = \min\{\text{new\_count}, \lceil d(v)/2 \rceil\} - \min\{\text{old\_count}, \lceil d(v)/2 \rceil\}$$

Questa strategia evita ricalcoli globali e consente di aggiornare solo i valori realmente interessati dalla modifica del seed set.

Inoltre, per selezionare il nodo con il miglior rapporto guadagno/costo, è stata utilizzata una struttura dati di tipo heap (coda di priorità binaria), che permette l'estrazione del massimo in tempo  $O(\log n)$ . Ogni nodo è associato a un valore marginale normalizzato rispetto al suo costo, e viene gestito attraverso un heap di coppie (guadagno/costo, nodo). Per evitare di rieseguire il calcolo su tutti i nodi a ogni iterazione, il valore marginale viene aggiornato solo per quei nodi che risultano effettivamente influenzati dall'ultimo nodo inserito nel seed set, cioè i nodi adiacenti ai vicini del nodo selezionato.

Infine, è stato previsto un meccanismo per la costruzione incrementale del seed set, permettendo di conservare il valore corrente della funzione submodulare e del numero di vicini attivati, così da riutilizzare le informazioni anche in esperimenti con budget crescenti, senza dover ripartire da zero.

Nel complesso, queste ottimizzazioni ci hanno consentito di dimezzare i tempi di esecuzione, mantenendo invariata la logica greedy e la qualità del risultato.

### 3.4 Weighted Target Set Selection(WTSS)

(Cordasco et al., 2015) L'algoritmo *Weighted Target Set Selection* (WTSS) è una variante pesata del problema *Target Set Selection* (TSS). Come descritto in Cordasco et al., 2018, l'obiettivo è individuare un insieme  $S$  di nodi del grafo  $G$  che costituisca un target set, minimizzando al contempo la somma dei pesi associati ai nodi selezionati. Più formalmente, il problema da risolvere può essere riformulato come segue.

*Dato un grafo  $G = (V, E)$ , una funzione di costo (o peso)  $c : V \rightarrow \mathbb{R}^+$ , e una funzione soglia  $t : V \rightarrow \mathbb{N}^+$  — che assegna a ciascun nodo  $v$  il numero minimo di vicini attivi necessari per la sua attivazione — l'obiettivo è determinare un seed set  $S \subseteq V$  che massimizzi l'influenza nel grafo, minimizzando al contempo la somma dei costi dei nodi appartenenti a  $S$ .*

La target set selection non solo è un problema NP-Hard ( $O(2^{\log^{(1-\epsilon)n}})$ ) ma non è nemmeno approssimabile in un tempo costante, per questo motivo in letteratura vengono proposte diverse euristiche per affrontarlo. Una di esse è rappresentata dall'Algoritmo 2 (WTSS) di seguito riportato.

La procedura **WTSS()** comincia inizializzando il *target set* di output  $S$  come insieme vuoto. L'algoritmo scorre i nodi del grafo  $G = (V, E)$ , valutando per ciascuno se debba essere inserito in  $S$  oppure rimosso definitivamente dal grafo. A tal fine, l'insieme dei nodi  $V$  viene copiato in un nuovo insieme  $U$ , che rappresenta i nodi ancora attivi nel processo.

Per ogni nodo  $v \in U$ , vengono definite e inizializzate le seguenti grandezze:

- $\delta(v)$ : grado di  $v$  rispetto al grafo indotto da  $U$ , inizialmente uguale al grado in  $G$ ;
- $k(v)$ : soglia di attivazione del nodo, inizializzata come  $k(v) = t(v)$ ;
- $N(v)$ : insieme dei vicini di  $v$  all'interno di  $U$ , inizialmente uguale a quello in  $G$ .

Queste strutture vengono aggiornate durante l'esecuzione per riflettere i cambiamenti nel grafo man mano che i nodi vengono rimossi da  $U$ . Un nodo può essere rimosso in due modi: o viene incluso nel seed set  $S$ , oppure viene scartato. Questa decisione si basa sull'analisi di tre casi distinti:

**Caso 1 – Soglia pari a zero** Si identificano i nodi  $v \in U$  per cui  $k(v) = 0$ . Tali nodi possono essere attivati direttamente dai nodi già esterni a  $U$  (ossia quelli già attivi o appartenenti a  $S$ ). Pertanto, non è necessario includerli nel seed set. Il nodo viene rimosso da  $U$ , e per ciascun vicino  $u \in N(v)$ , la soglia di attivazione viene aggiornata come:

$$k(u) \leftarrow \max(0, k(u) - 1)$$

**Caso 2 – Nodo non attivabile dai vicini rimasti** Se esiste un nodo  $v \in U$  tale che  $\delta(v) < k(v)$ , significa che il numero di vicini ancora presenti in  $U$  non è sufficiente per

---

**Algorithm 2** WTSS( $G$ )

---

**Input:** A graph  $G = (V, E)$  with thresholds  $t(v)$  and costs  $c(v)$ , for  $v \in V$ .  
**Output:** A target set  $S$  for  $G$ .

```

1:  $S = \emptyset; U = V$ 
2: for each  $v \in V$  do
3:    $\{\delta(v) = d_G(v); k(v) = t(v); N(v) = \Gamma_G(v)\}$ 
4:
5: while  $U \neq \emptyset$  do
6:   /* Select one vertex and eliminate it from the graph as specified in the following cases */
7:   if there exists  $v \in U$  s.t.  $k(v) = 0$  then
8:     /* Case 1: The selected vertex  $v$  is activated by the influence of its
9:        neighbors in  $V - U$  only; it can then influence its neighbors in  $U$  */
10:    for each  $u \in N(v)$  do
11:       $k(u) = \max\{0, k(u) - 1\}$ 
12:
13:   else
14:     if there exists  $v \in U$  s.t.  $\delta(v) < k(v)$  then
15:       /* Case 2: The vertex  $v$  is added to  $S$ , since no sufficient neighbors
16:          remain in  $U$  to activate it;  $v$  can then influence its neighbors in  $U$  */
17:        $S = S \cup \{v\}$ 
18:       for each  $u \in N(v)$  do
19:          $k(u) = k(u) - 1$ 
20:
21:     else
22:       /* Case 3: The selected vertex  $v$  will be activated by its neighbors in  $U$  */
23:        $v = \arg \max_{u \in U} \left\{ \frac{c(u) \cdot k(u)}{\delta(u)(\delta(u) + 1)} \right\}$ 
24:
25:
26:   /* Remove the selected vertex  $v$  from the graph */
27:   for each  $u \in N(v)$  do
28:      $\{\delta(u) = \delta(u) - 1; N(u) = N(u) - \{v\}\}$ 
29:
30:    $U = U - \{v\}$ 
31:
```

---

soddisfare la soglia di attivazione. In tal caso,  $v$  viene aggiunto a  $S$ , e per ogni  $u \in N(v)$ , la soglia viene aggiornata come:

$$k(u) \leftarrow k(u) - 1$$

Poiché il caso 1 viene sempre verificato prima, si ha la garanzia che  $k(u) \geq 1$ , evitando soglie negative.

**Caso 3 – Rimozione guidata da funzione di priorità** Se non si verificano i casi precedenti, si seleziona un nodo influenzabile dai suoi vicini rimasti in  $U$ . Per ogni  $u \in U$ , si calcola il valore:

$$\frac{c(u) \cdot k(u)}{\delta(u)(\delta(u) - 1)}$$

e si seleziona il nodo  $v$  che massimizza tale espressione:

$$v = \arg \max_{u \in U} \left\{ \frac{c(u) \cdot k(u)}{\delta(u)(\delta(u) - 1)} \right\}$$

Il nodo  $v$  viene rimosso da  $U$  e, per ogni  $u \in N(v)$ , si aggiornano:

$$\delta(u) \leftarrow \delta(u) - 1, \quad N(u) \leftarrow N(u) \setminus \{v\}$$

L'algoritmo **WTSS()** risulta il più veloce tra quelli considerati. Poiché nei casi 1 e 3 vengono eliminati nodi senza includerli nel seed set, è possibile ottenere un *upper bound* alla dimensione massima di  $S$ . Questo consente di interrompere l'incremento del budget una volta raggiunta tale soglia, riducendo il numero di iterazioni rispetto ad altri algoritmi.

*Nota:* la dimensione del seed set  $S$  è strettamente minore di  $|V|$  fintanto che esiste almeno un arco nel grafo  $G$ . Se invece  $G$  è un insieme di nodi isolati, allora  $|S| = |V|$ .

All'algoritmo originale è stata introdotta una modifica per garantire che il costo complessivo dei nodi selezionati non superi il valore del budget specificato come parametro di input. In particolare, ogni volta che un nuovo nodo viene candidato per l'inclusione nel seed set  $S$ , si verifica che la sua aggiunta mantenga il costo totale entro i limiti del budget assegnato. Se questa condizione è soddisfatta, il nodo viene effettivamente inserito in  $S$ ; in caso contrario, il nodo viene semplicemente rimosso dal grafo e l'algoritmo continua con i nodi rimanenti.

L'esecuzione termina quando uno dei due criteri di arresto è soddisfatto:

- l'insieme  $U$  risulta vuoto, ovvero tutti i nodi sono stati processati;
- la somma dei costi dei nodi in  $S$  raggiunge esattamente il budget  $k$ .

### 3.5 SMiLe-CoDe

L'Algoritmo 3, denominato *SMiLe-CoDe*, è una strategia euristica per la selezione di un insieme di nodi seed all'interno di un grafo  $G = (V, E)$ , che combina una partizione del grafo in comunità con una selezione greedy basata sulla centralità. L'obiettivo, anche in questo caso, è massimizzare la diffusione dell'informazione rispettando un vincolo di budget  $k$ , attraverso una distribuzione locale del budget tra le comunità identificate nel grafo.

In una prima fase, l'algoritmo esegue una partizione del grafo mediante il metodo di Louvain (Blondel et al., 2008), ottenendo un insieme disgiunto di comunità  $C = \{C_1, C_2, \dots, C_m\}$ . A ciascuna comunità  $C_i$  viene assegnato un budget  $k_i$  proporzionale alla sua dimensione, ovvero:

$$k_i = \left\lfloor k \cdot \frac{|C_i|}{|V|} \right\rfloor$$

Successivamente, per ogni comunità  $C_i$ , si seleziona un sottoinsieme di nodi  $S_i \subseteq C_i$  tramite una procedura greedy basata sulla centralità tra le più rilevanti nel contesto (betweenness centrality). L'inclusione di un nodo nel seed set locale avviene solo se il suo costo  $c(v)$  (basato su un attributo specifico) non eccede il budget rimanente per la comunità. L'insieme finale di seed è ottenuto dall'unione dei sottoinsiemi locali:

$$S = \bigcup_{i=1}^m S_i$$

**Gestione del budget residuo** Dopo la selezione locale dei seed nelle singole comunità, potrebbe rimanere una parte del budget inutilizzata (ad esempio a causa di costi troppo elevati rispetto al budget disponibile in una comunità). Per evitare sprechi, SMiLe-CoDe impiega una fase globale finale in cui si selezionano ulteriori nodi dal grafo intero.

Tale selezione globale viene effettuata ordinando tutti i nodi rimanenti (non ancora scelti come seed) secondo la centralità e includendo quelli il cui costo non eccede il budget residuo. Il processo continua in modo greedy finché il budget viene completamente speso o non ci sono più candidati economicamente sostenibili.

**Schema complessivo** Pertanto, il seed set finale è dato da:

$$S = \left( \bigcup_{i=1}^m S_i \right) \cup S_{\text{globale}}$$

dove  $S_i$  è il sottoinsieme di seed selezionati nella comunità  $C_i$  e  $S_{\text{globale}}$  è l'insieme di nodi scelti nella fase di selezione globale.

**Strategia di selezione** La selezione all'interno di ciascuna comunità avviene secondo una strategia greedy deterministica, ordinando i nodi in base a una metrica di centralità (default: betweenness centrality) precalcolata e assegnata come attributo. L'algoritmo itera su tale ordinamento, includendo un nodo nel seed set locale se il suo costo non eccede il budget disponibile.

SMiLe-CoDe sfrutta quindi la struttura comunitaria del grafo per ottenere una copertura più equilibrata e distribuita tra diverse regioni della rete, evitando concentrazioni di seed in aree altamente connesse. Inoltre, la separazione del problema in sottoproblemi locali riduce la complessità computazionale complessiva, rendendo l'approccio adatto a grafi di grandi dimensioni.

---

**Algorithm 3** SMiLe-CoDe( $G, k, c, \text{centrality}$ )

---

**Input:** Grafo  $G = (V, E)$ , budget totale  $k$ , funzione di costo  $c(v)$ , misura di centralità  
**Output:** Seed set  $S$

- 1: Calcola partizione in comunità  $C_1, \dots, C_m$  con Louvain
  - 2: Assegna budget  $k_i = \lfloor k \cdot \frac{|C_i|}{|V|} \rfloor$
  - 3: Inizializza  $S = \emptyset$ ,  $k_{\text{residuo}} \leftarrow k$
  - 4: **for** ogni  $C_i$  **do**
  - 5:   Ordina i nodi in base a centralità decrescente
  - 6:   Seleziona greedy i nodi con costo compatibile con  $k_i$
  - 7:   Aggiorna  $S, k_{\text{residuo}}$
  - 8:
  - 9: **if**  $k_{\text{residuo}} > 0$  **then**
  - 10:   Ordina i nodi non selezionati per centralità crescente
  - 11:   Seleziona globalmente nodi finché il budget residuo lo consente
  - 12:   Aggiorna  $S$
  - 13:
  - 14: **return**  $S$
- 

### 3.5.1 Variante SMiLe-CoDe\_bridges

SMiLe-CoDe\_bridges è una variante di SMiLe-CoDe che utilizza i local bridge per la costruzione del seed set. L'idea dietro lo SMiLe-CoDe è quella che le reti sociali sono solitamente molto dense di collegamenti all'interno dei cluster ma poco dense tra i cluster, di conseguenza potrebbe essere favorevole influenzare nodi che sono centrali all'interno dei cluster per migliorare la probabilità di influenzare l'intera comunità. Questa scelta dei nodi viene svolta all'interno di ogni comunità come descritta precedentemente. Il problema dello SMiLe-CoDe è che si concentra troppo sui nodi centrali all'interno delle comunità e questo può portare ad una scelta di nodi che non è in grado di coprire tutte le comunità se queste sono particolarmente isolate. Per aumentare l'influenza tra i diversi cluster si è pensato di utilizzare il budget residuo per selezionare i nodi che fanno parte dei local bridge. Questi nodi vengono sottoposti allo stesso processo greedy basato sulla centralità della prima fase di SMiLe di modo che vengano presi prima i nodi che sono più importanti all'interno delle comunità. Dall'analisi della rete è emerso che la maggior parte dei *local bridge* collega nodi che, in assenza di tali archi, risulterebbero isolati. Questa proprietà è particolarmente utile, poiché ci permette di garantire l'attivazione di questi nodi periferici e, di conseguenza, di estendere l'influenza anche alle aree più marginali delle comunità.

## 3.6 Diffusione dell'influenza - Majority cascade

Per analizzare la propagazione dell'influenza all'interno del grafo considerato, è stato adottato un modello di diffusione basato su una dinamica a soglia di maggioranza, riportato nell'Algoritmo 4. In questo contesto, la diffusione parte da un insieme iniziale di nodi attivi (*seed set*), selezionati a partire dai risultati ottenuti con i tre algoritmi

precedentemente descritti. L'influenza si propaga iterativamente nel grafo seguendo la regola per cui un nodo inattivo diventa attivo se almeno la metà dei suoi vicini è già stato influenzato nel passo precedente.

Il processo iterativo prosegue fino al raggiungimento di uno stato stazionario, ovvero quando l'insieme dei nodi influenzati non varia più tra due iterazioni successive. Durante ogni iterazione, vengono valutati tutti i nodi non ancora attivi, tenendo conto del numero di vicini già attivati, e si aggiorna l'insieme complessivo dei nodi influenzati. Per ogni scenario sperimentale, definito da un differente insieme iniziale di nodi attivi, è stata monitorata l'evoluzione del processo di diffusione, registrando sia il numero complessivo di nodi influenzati al termine della propagazione sia il numero di iterazioni necessarie per raggiungere la stabilizzazione. Inoltre, sono stati raccolti i tempi di esecuzione per ciascun esperimento, al fine di valutare l'efficienza computazionale del processo su grafi di dimensioni reali.

---

**Algorithm 4** Majority Cascade
 

---

**Input:** Grafo  $G = (V, E)$ , seed set iniziale  $S \subseteq V$

**Output:** Insieme finale di nodi influenzati  $I$ , numero di round  $r$

```

1:  $I = S$  {Insieme dei nodi influenzati}
2:  $I_{\text{prev}} = \emptyset$ 
3:  $r = 0$ 
4: while  $I \neq I_{\text{prev}}$  do
5:    $r = r + 1$ 
6:    $I_{\text{prev}} = I$ 
7:    $N = \emptyset$  {Nuovi nodi influenzati in questo round}
8:   for all  $v \in V \setminus I_{\text{prev}}$  do
9:      $N_v = \{u \in \text{Neighbors}(v) : u \in I_{\text{prev}}\}$ 
10:    if  $|N_v| \geq \left\lceil \frac{|N_v|}{2} \right\rceil$  then
11:       $N = N \cup \{v\}$ 
12:
13:
14:    $I = I \cup N$ 
15:
16:
17: return  $I, r$ 
  
```

---

# 4

## Risultati ottenuti

In questo capitolo vengono analizzate in dettaglio le capacità dei diversi algoritmi di massimizzare la diffusione dell'influenza all'interno della rete *ego-Facebook* in esame, a fronte di vincoli di budget variabili. L'indagine sperimentale si è concentrata su quattro algoritmi — *CSG* (ottimizzato), *WTSS*, *SMiLe-CoDe* e la sua variante *bridges* — nel massimizzare l'influenza in reti complesse, valutandone l'efficacia in relazione al budget disponibile e a tre diverse funzioni di costo: *Cost1* (grado inverso), *Cost2* (casuale), e *Cost3* (centralità di betweenness normalizzata).

L'analisi si articola in tre direzioni: il confronto tra i diversi algoritmi rispetto alla stessa funzione di costo, la sensibilità di ciascun approccio al variare di essa, e infine la valutazione dei tempi computazionali, con particolare attenzione alla ricerca del seed set e alla simulazione della diffusione.

### 4.1 Analisi della diffusione dell'influenza

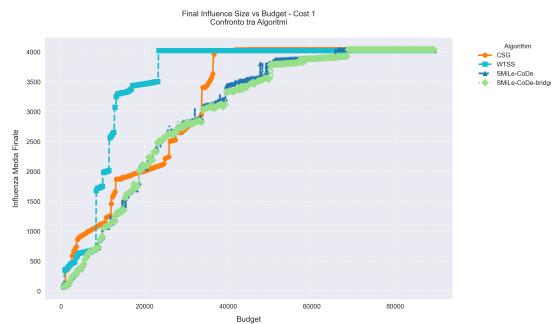
Nel seguito, si presentano due analisi complementari: nella prima, si confrontano le prestazioni dei vari algoritmi per ciascuna funzione di costo; nella seconda, si approfondisce la sensibilità di ciascun algoritmo rispetto al variare della funzione di costo utilizzata.

#### 4.1.1 Prestazioni comparative degli algoritmi al variare della funzione di costo

Nel caso della funzione *Cost1* (Figura 4.1), emerge chiaramente la superiorità dell'algoritmo **WTSS**, che riesce a raggiungere una diffusione pressoché completa già con budget contenuti, dimostrando un'elevata efficienza anche in scenari con risorse limitate.

**CSG** evidenzia una crescita più regolare e progressiva dell'influenza rispetto al budget, mantenendo comunque buone performance e raggiungendo gradualmente valori prossimi alla copertura totale.

Gli algoritmi **SMiLe-CoDe** e **SMiLe-CoDe-bridges**, pur partendo da livelli inferiori di influenza, mostrano una crescita costante che consente loro di colmare parzialmente il divario con i metodi più performanti all'aumentare del budget. L'andamento delle curve è quasi sovrapponibile per i due algoritmi, con alcune lievi divergenze osservabili tra i 40 000 e i 60 000 unità di budget, dove *SMiLe-CoDe* tende a ottenere una diffusione leggermente maggiore. Un comportamento interessante emerge nella fascia tra 20 000 e 35 000 unità, in cui entrambi gli algoritmi superano temporaneamente **CSG**, suggerendo una buona capacità di propagazione in scenari intermedi.



**Figura 4.1:** Prestazioni comparative degli algoritmi con funzione di costo 1

Nel caso della funzione *Cost2* (Figura 4.2), caratterizzata da un'assegnazione casuale dei costi, l'andamento complessivo cambia significativamente. **WTSS** conferma un'elevata efficienza nelle prime fasi, ma viene raggiunto da **CSG** a budget leggermente più alti. Entrambi mostrano una buona capacità di adattamento alla variabilità dei costi. Le due versioni dell'algoritmo **SMiLe-CoDe** presentano un comportamento quasi identico anche in questa configurazione, con curve che crescono in modo graduale e senza brusche variazioni, indicando una maggiore stabilità rispetto ai cambiamenti nella distribuzione dei costi.

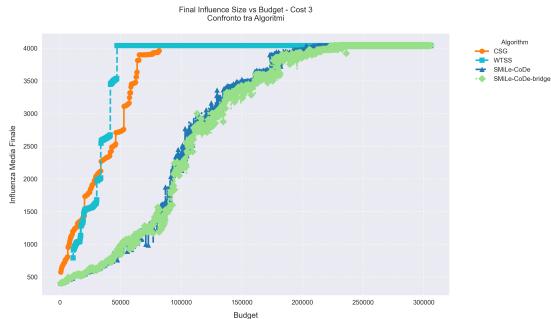


**Figura 4.2:** Prestazioni comparative degli algoritmi con funzione di costo 2

Nel contesto della funzione *Cost3* (Figura 4.3), basata sulla centralità, **WTSS** si conferma particolarmente efficace, raggiungendo rapidamente la convergenza anche in presenza di costi più eterogenei. **CSG** segue un andamento più graduale ma rimane competitivo, senza presentare oscillazioni marcate.

Tra le due versioni di **SMiLe-CoDe**, quella standard riesce a ottenere risultati leggermente migliori per la maggior parte dei budget. Tuttavia, entrambe restano meno per-

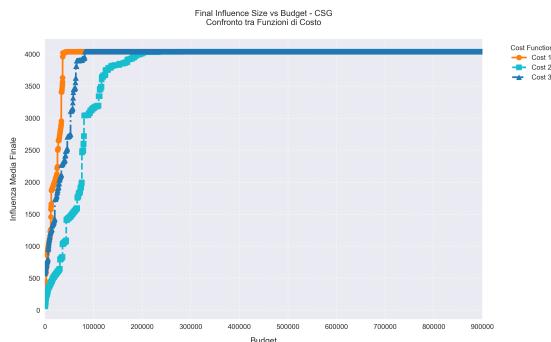
formanti rispetto a WTSS e CSG, confermando un comportamento più penalizzato in presenza di costi basati su misure globali di centralità.



**Figura 4.3:** Prestazioni comparative degli algoritmi con funzione di costo 3

#### 4.1.2 Sensibilità degli algoritmi alle diverse funzioni di costo

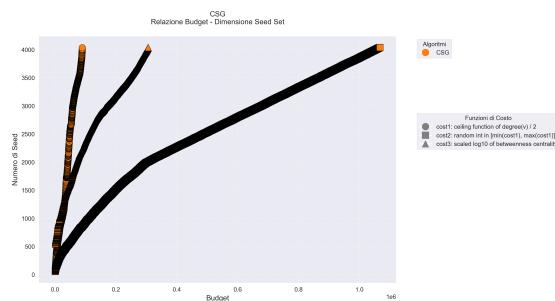
Nel valutare la sensibilità degli algoritmi al variare della funzione di costo, si osserva che **CSG** mantiene un comportamento stabile e coerente. Le tre curve corrispondenti a *Cost1*, *Cost2* e *Cost3* (Figura 4.4) risultano infatti molto simili, suggerendo una robustezza strutturale dell'algoritmo rispetto alla natura del costo assegnato ai nodi. La funzione *Cost1* consente comunque di ottenere le migliori performance, evidenziando una lieve superiorità in termini di influenza media raggiunta.



**Figura 4.4:** Sensibilità di CSG alle funzioni di costo

Queste osservazioni trovano ulteriore conferma nell'analisi del numero di seed selezionati in relazione all'incremento del budget. Come mostrato in Figura 4.5, la funzione di costo *Cost1* permette di includere un numero significativamente maggiore di nodi seed già a budget contenuti, mostrando una crescita rapida. La funzione *Cost3* segue un comportamento intermedio, mentre *Cost2* si rivela la meno efficiente, consentendo la selezione di un numero limitato di seed anche in presenza di budget più elevati.

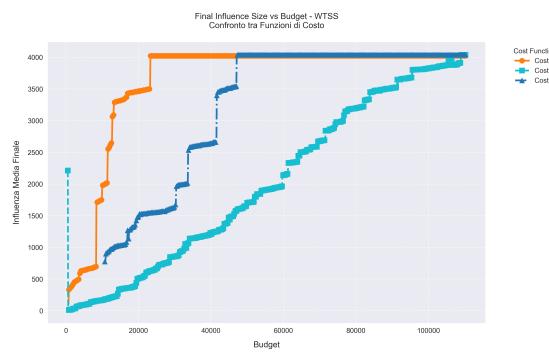
Per quanto riguarda **WTSS**, si nota un comportamento particolarmente efficiente: il grafico (Figura 4.6) è l'unico ad essere rappresentato su un intervallo ridotto [0, 100 000], a testimonianza del fatto che l'algoritmo raggiunge la copertura totale dei nodi in tempi molto più rapidi rispetto agli altri metodi. In particolare, con la funzione *Cost1* viene raggiunta la convergenza con un budget di poco superiore a 20 000, mentre *Cost3* con-



**Figura 4.5:** Evoluzione della dimensione del seed set in funzione del budget nell'algoritmo CSG

sente di ottenere risultati simili con un budget intorno ai 40 000. La funzione *Cost2*, al contrario, genera una crescita più lineare e meno efficace.

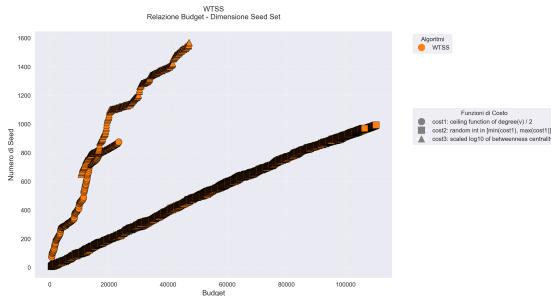
Un'osservazione interessante emerge dalla presenza di un picco anomalo nella curva relativa a *Cost2*, localizzato a budget molto bassi. Questo comportamento potrebbe indicare l'attivazione precoce di un nodo strategico — come un hub intercomunitario — che innesca un'amplificazione iniziale dell'influenza, seguita però da una diminuzione di efficienza dovuta alla selezione di nodi marginali con minore impatto sulla diffusione.



**Figura 4.6:** Sensibilità di WTSS alle funzioni di costo

Anche per questo algoritmo, queste osservazioni trovano ulteriore conferma nell'analisi del numero di seed selezionati in relazione all'incremento del budget. Come mostrato in Figura 4.7, la funzione di costo *Cost1* permette di includere un numero significativamente maggiore di nodi seed già a budget contenuti, mostrando una crescita molto rapida che conduce alla copertura completa intorno a 20 000 unità. La funzione di costo *Cost3* presenta una pendenza più moderata che raggiunge la convergenza solo intorno a 40 000 di budget. Infine, la funzione di costo *Cost2*, anche in questo caso, si rivela la meno efficiente: dopo un picco iniziale dovuto probabilmente alla selezione fortunata di un hub, la crescita diventa lineare e richiede un budget molto più elevato per arrivare alla copertura totale, evidenziando una maggiore variabilità quando il costo non riflette alcuna struttura topologica.

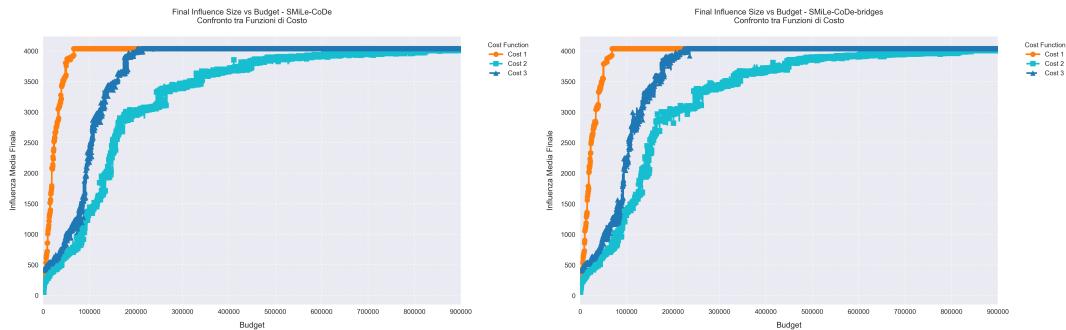
Gli algoritmi **SMiLe-CoDe** e **SMiLe-CoDe-bridges** sono visualizzati congiuntamente in Figura 4.8 per mettere in luce la loro elevata somiglianza. In entrambi i casi, la funzione *Cost1* risulta la più vantaggiosa, consentendo la convergenza con un budget inferiore



**Figura 4.7:** Evoluzione della dimensione del seed set in funzione del budget nell'algoritmo WTSS

alle 100 000 unità. Seguono *Cost3* e infine *Cost2*, che richiede il maggior dispendio di risorse per ottenere una diffusione completa.

La quasi totale sovrapposizione tra le due curve suggerisce che, nella maggior parte dei casi, i nodi identificati come bridge nella versione estesa dell'algoritmo abbiano costi troppo elevati per essere selezionati nella fase finale di allocazione del budget. Di conseguenza, *SMiLe-CoDe-bridges* finisce per comportarsi in modo quasi identico alla sua versione standard, non riuscendo a sfruttare appieno il contributo topologico offerto dai bridge tra comunità. Questo può essere dovuto alla struttura della rete presa in esame. Nella Sezione 2.3 è mostrato come la quasi totalità dei local bridge siano in realtà dei bridge che connettono nodi isolati alla componente connessa. Siccome **SMiLe-CoDe-bridges** inserisce entrambi i nodi del bridge in  $S$ , non permette di ottenere dall'aggiunta dei bridge il risultato sperato (inserisce nodi che sarebbero influenzati comunque). Un possibile miglioramento potrebbe essere quello di considerare solo il nodo del ponte con centralità maggiore invece che entrambi.



**Figura 4.8:** Sensibilità di SMiLe-CoDe e SMiLe-CoDe-bridges alle funzioni di costo

Anche per SMiLe-CoDe e SMiLe-CoDe-bridges, la relazione tra budget e dimensione del seed set (Figura 4.9) conferma la gerarchia tra le tre funzioni di costo. La curva relativa a *Cost1* presenta la pendenza più ripida, consentendo di selezionare rapidamente un elevato numero di nodi seed già con budget dell'ordine delle decine di migliaia. La funzione *Cost3* mantiene un andamento più graduale e privo di picchi, segno di una scelta progressiva dei nodi ponte guidata dalla centralità di betweenness. Infine, l'assegnazione casuale (*Cost2*) impone la necessità di budget molto più elevati (spesso superiori alle centinaia di migliaia) per arrivare a copertura completa, a causa dell'assenza di correlazione tra costo e potere di diffusione dei nodi. La versione "bridges"

(riportata a destra in Figura 4.9) di SMiLe-CoDe conferma in pieno il pattern osservato per l'algoritmo standard. Ciò conferma che, come affermato in precedenza, nel grafo analizzato, i bridge non forniscono un vantaggio competitivo significativo rispetto ai nodi scelti dalla versione base dell'algoritmo.



**Figura 4.9:** Evoluzione della dimensione del seed set in funzione del budget negli algoritmi SMiLe-CoDe e SMiLe-CoDe-bridges

## 4.2 Risultati temporali

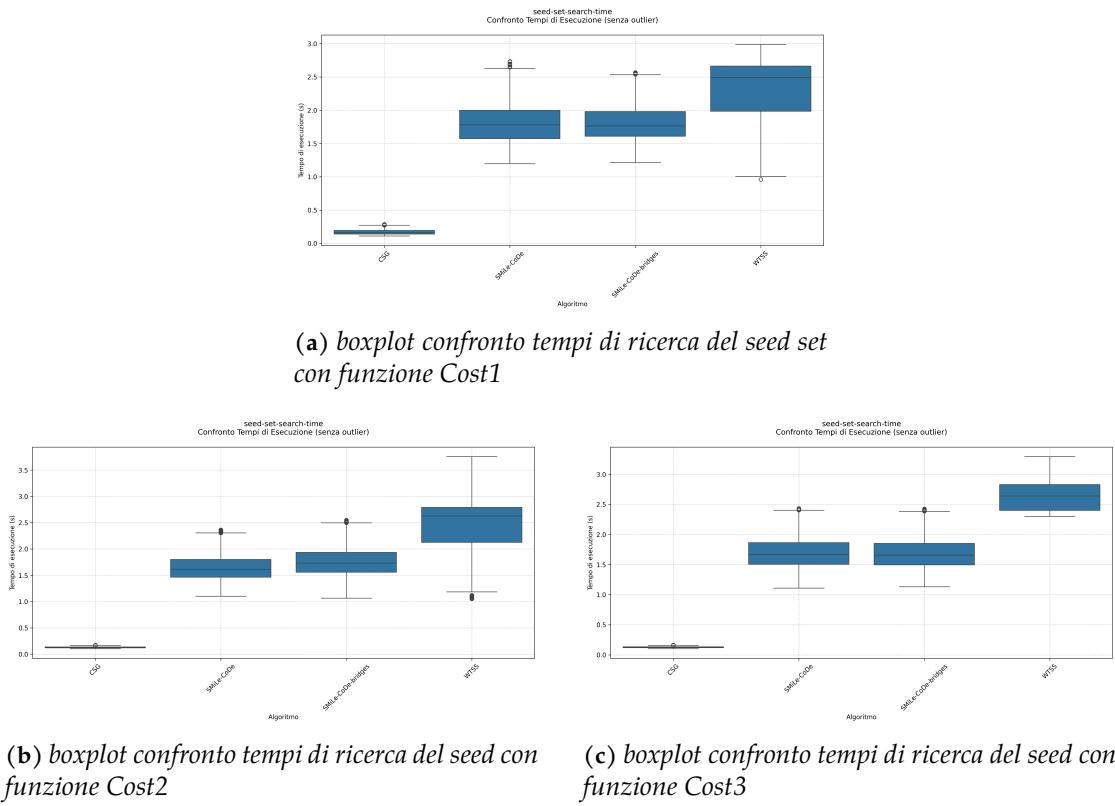
Durante le esecuzioni degli algoritmi studiati e delle iterazioni di *influence cascade* dei seed set, sono stati salvati i tempi di esecuzione per poter fare un confronto temporale tra i diversi algoritmi. È stata utilizzata la libreria `time` di Python per tenere traccia delle istanze temporali, le quali sono servite per le analisi di seguito riportate.

### 4.2.1 Confronto algoritmi nella ricerca del seed set

Per gli algoritmi di ricerca dei seed set, si definisce come *istanza temporale* il tempo necessario per completare la ricerca di un insieme  $S$  tale che  $c(S) \leq k$ , dove  $k$  rappresenta il budget disponibile.

In primo luogo, dai boxplot in Figura 4.10 si può osservare il tempo impiegato dalle singole iterazioni per il calcolo del seed set. L'algoritmo **WTSS** presenta i tempi di ricerca più elevati, con un valore mediano di circa 2,5 secondi per tutte e tre le funzioni di costo considerate. Seguono **SMiLe-CoDe-bridges** e **SMiLe-CoDe**, i cui tempi mediani si attestano tra 1,65 e 1,75 secondi. Infine, **CSG** risulta essere l'algoritmo più efficiente in termini di tempo per iterazione, con valori che raramente superano i 0,25 secondi.

Si osserva inoltre che l'algoritmo **WTSS** presenta una maggiore variabilità nei tempi di esecuzione delle iterazioni per le funzioni di costo **Cost1** e **Cost2** (che operano sul medesimo intervallo di valori), mentre mostra una distribuzione più compatta con la funzione di costo basata sulla centralità. I due algoritmi **SMiLe**, invece, presentano comportamenti pressoché identici e, analogamente al **CSG**, mostrano valori di tempo molto simili per tutte e tre le tipologie di costo. È rilevante notare che le diverse funzioni di costo non influenzano in modo significativo i tempi di esecuzione: l'unica eccezione è rappresentata da un lieve incremento (circa 0,5 secondi) del quarto quartile nel caso della funzione randomica.



**Figura 4.10:** Confronto tempi di ricerca del seed set con diverse funzioni di costo

Si ricorda che la ricerca dei seed set è stata effettuata su tre diverse funzioni di costo, al variare del budget. Tuttavia, il tempo medio di esecuzione per singola iterazione non è rappresentativo del tempo complessivo richiesto da ciascun algoritmo per esplorare l'intero intervallo di budget considerato. In Figura 4.11 sono riportati gli istogrammi che mostrano il tempo totale trascorso da ciascun algoritmo in tutte le iterazioni, per ciascuna funzione di costo.

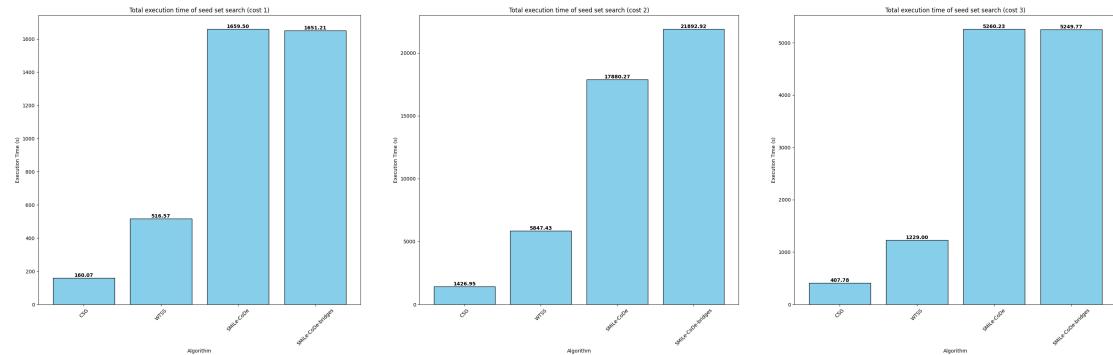
Sebbene **WTSS** sia l'algoritmo con il tempo medio più elevato per singola iterazione, il suo tempo di esecuzione complessivo risulta inferiore rispetto a quello dei due algoritmi **SMiLe**. È interessante osservare come la funzione **Cost2**, che assegna valori distribuiti uniformemente ai nodi del grafo, comporti un notevole aumento del tempo di calcolo, richiedendo più iterazioni per stabilizzare il target set.

Nel caso di **CSG**, ad ogni iterazione il nuovo seed set viene costruito a partire da quello dell'iterazione precedente (con budget aggiornato), mentre in **WTSS** l'algoritmo può terminare anticipatamente una volta raggiunta una taglia massima  $|S|$ , escludendo i nodi facilmente influenzabili. Queste ottimizzazioni contribuiscono a contenere il tempo totale di esecuzione, rendendo **CSG** e **WTSS** sensibilmente più rapidi rispetto agli algoritmi **SMiLe**.

Nonostante ciò, entrambi gli algoritmi impiegano circa 10 volte il tempo su **Cost2** rispetto a **Cost1**, e circa 3,5 volte in più rispetto a **Cost3**. Lo stesso rapporto è riscontrabile anche per **SMiLe-CoDe**, mentre per **SMiLe-CoDe-bridges** il tempo di esecuzione con la funzione randomica è circa 13 volte superiore a quello con **Cost1**, e 4 volte rispetto

alla funzione basata sulla centralità (**Cost3**).

È opportuno evidenziare che ai due algoritmi **SMiLe** non sono state applicate ottimizzazioni analoghe a quelle adottate in **CSG**, motivo per cui i loro tempi di esecuzione risultano significativamente più elevati. Questo è attribuibile anche al fatto che, come per **CSG**, all'aumentare del budget gli algoritmi possono continuare ad aggiungere nodi al seed set  $S$  fino a raggiungere l'intero insieme  $S = V$ .



(a) istogramma confronto tempo di ricerca dei seed set con Cost1    (b) istogramma confronto tempo di ricerca dei seed set con Cost2    (c) istogramma confronto tempo di ricerca dei seed set con Cost3

**Figura 4.11:** Confronto tempi di ricerca del seed set per diversi tipi di costo al variare del budget

#### 4.2.2 Confronto tempo di esecuzione dei cascade

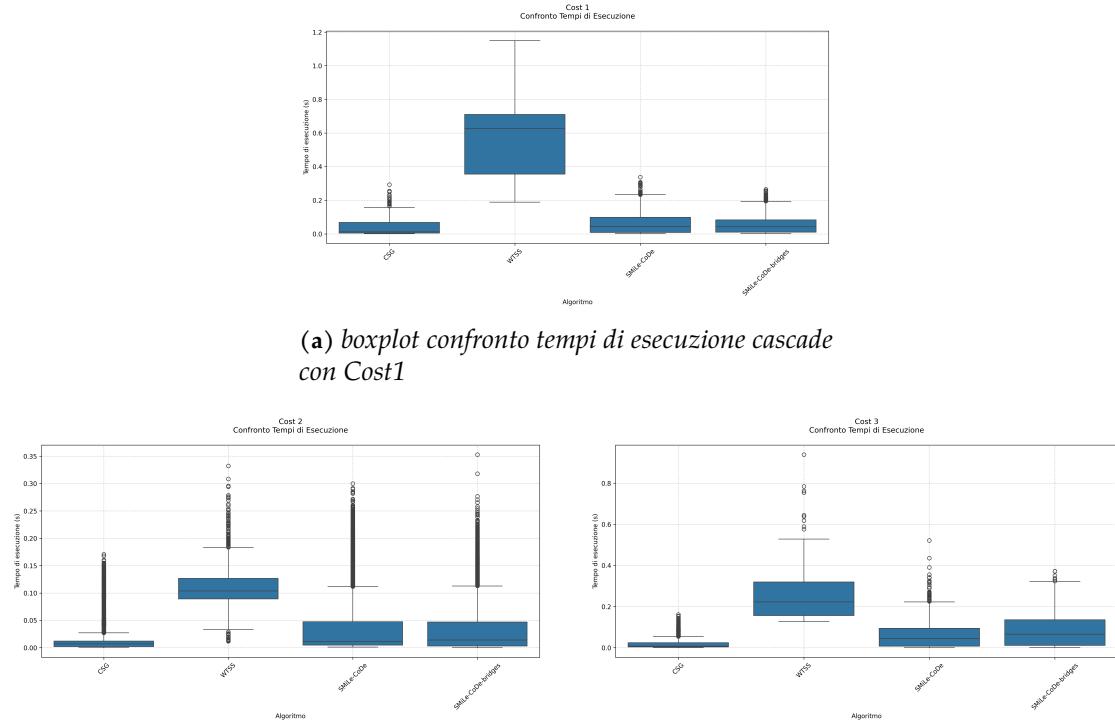
Per valutare i tempi di esecuzione dei processi di diffusione (cascade), è stato considerato il tempo necessario affinché un determinato seed set completi il proprio processo di influenza. Ogni valore temporale è stato etichettato in base all'algoritmo che ha generato il seed set, poiché la procedura di valutazione dell'influenza è stata mantenuta identica per tutti i casi.

Come evidenziato in Figura 4.12, anche nelle simulazioni dei cascade l'algoritmo **WTSS** risulta essere quello con i tempi di esecuzione più elevati. Tuttavia, questo non rappresenta necessariamente un aspetto negativo: **WTSS**, infatti, è l'algoritmo più efficace in termini di influenza totale, e impiega semplicemente un numero maggiore di round per completare la diffusione, in quanto l'influenza tende a propagarsi più a lungo.

Un aspetto interessante riguarda l'effetto della funzione di costo utilizzata per la selezione del seed set sul tempo complessivo del cascade. I seed set generati sulla base della funzione **Cost2** portano a processi di diffusione più brevi rispetto a quelli basati su **Cost3** e **Cost1**. Ciò è probabilmente dovuto alla minore efficacia dei seed set selezionati con **Cost2**, che portano ad un arresto più rapido del processo di influenza. Al contrario, la funzione **Cost1** consente la generazione di seed set che producono processi di diffusione più duraturi e, generalmente, più ampi.

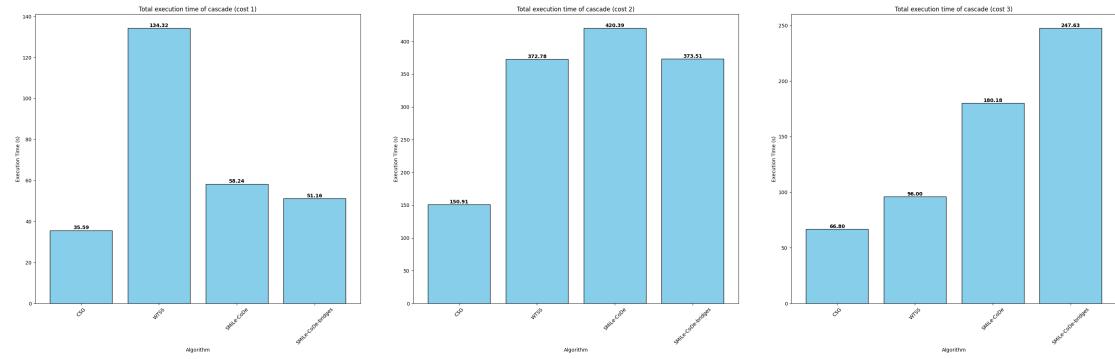
È importante sottolineare che un numero maggiore di round non implica necessariamente una migliore capacità di diffusione. Tuttavia, nel caso di reti poco o moderatamente dense, come quella analizzata, una diffusione più prolungata è spesso indi-

ce di una maggiore copertura, poiché i collegamenti sono più diradati e l'influenza si propaga più lentamente.



**Figura 4.12:** Confronto tempi di esecuzione cascade per diversi tipi di costo

Analizzando il tempo totale delle esecuzione dei diversi cascade sui seed set generati al variare del budget è possibile notare che la scelta della funzione di costo è stata molto conseguenziale per il tempo richiesto nel processo di influenza. In generale tutti e quattro gli algoritmi hanno avuto tempi più ristretti con la funzione Cost1 e più lunghi con Cost2. Con l'analisi fatta precedentemente sull'influenza, questo può suggerire che i seed scelti con un funzione randomica (Cost2) siano di qualità più bassa di quelli scelti con le altre funzioni di costo.



**Figura 4.13:** Confronto tempi di esecuzione cascade per diversi tipi di costo al variare del budget

# 5

## Conclusioni

Il presente studio ha evidenziato come la massimizzazione dell'influenza nelle reti sociali richieda un equilibrio tra efficienza computazionale, adattabilità ai vincoli di costo e comprensione della struttura della rete. I quattro algoritmi analizzati – CSG, WTSS, *SMiLe-CoDe* e la sua variante *bridges* – si sono comportati in modo differente, mostrando punti di forza e debolezza che riflettono approcci progettuali distinti.

WTSS si è rivelato il più efficace in condizioni favorevoli, in particolare quando i costi erano coerenti con la topologia locale (*Cost1*). In questi casi, è riuscito a raggiungere un'ampia diffusione anche con budget limitati, grazie a una selezione mirata dei nodi più strategici. Tuttavia, la sua sensibilità a funzioni di costo irregolari (*Cost2*) ha evidenziato una certa instabilità, che potrebbe limitarne l'affidabilità in scenari più realistici e variabili.

CSG, pur non essendo il più performante in termini di influenza assoluta, ha mostrato stabilità e prevedibilità, con curve di diffusione regolari.

Infine, gli approcci basati su comunità, *SMiLe-CoDe* e la sua variante *bridges*, hanno garantito una diffusione più bilanciata tra le diverse aree della rete, ma con alcuni limiti evidenti. In particolare, la variante che include i nodi ponte non ha prodotto un miglioramento tangibile, probabilmente a causa dei costi elevati associati a tali nodi.

Le funzioni di costo si sono rivelate determinanti nel definire il comportamento degli algoritmi. La funzione *Cost1*, basata sul grado dei nodi, ha favorito tutti gli approcci, in quanto ben allineata alla struttura locale della rete. Al contrario, *Cost2*, con assegnazione casuale, ha penalizzato le soluzioni meno flessibili, rivelando una maggiore dipendenza da pattern strutturali. Infine, *Cost3*, basata sulla centralità, ha avvantaggiato gli algoritmi in grado di sfruttare informazioni globali, come WTSS.

Guardando al futuro, sarebbe interessante esplorare *strategie ibride* che combinino i vantaggi di diversi approcci: la selettività di WTSS e la copertura omogenea di *SMiLe-CoDe*. Una tale combinazione potrebbe favorire una selezione più mirata dei nodi, eliminando quelli meno promettenti e migliorando al contempo sia l'efficienza computazionale sia la qualità del seed set ottenuto. Inoltre, l'adozione di *funzioni di costo dinamiche*, che evolvano con lo stato della diffusione, potrebbe migliorare la resilienza a comporta-

menti anomali. La *valorizzazione dei bridge* merita ulteriori indagini, soprattutto in reti con comunità scarsamente connesse. Infine, per affrontare problemi su larga scala, si potrebbe pensare di *parallelizzare le fasi più onerose*, come il rilevamento delle comunità e la selezione dei seed.

In conclusione, WTSS rappresenta la scelta più indicata in condizioni ideali di coerenza topologica e vincoli di budget. Gli approcci community-based si confermano promettenti, ma necessitano di ulteriori sviluppi per competere efficacemente.



# Bibliografia

- Blondel, Vincent D et al. (ott. 2008). «Fast unfolding of communities in large networks». In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10, P10008. issn: 1742-5468. doi: [10.1088/1742-5468/2008/10/p10008](https://doi.org/10.1088/1742-5468/2008/10/p10008). url: <http://dx.doi.org/10.1088/1742-5468/2008/10/P10008>.
- Cabrera-Martínez, Abel (2022). «New bounds on the double domination number of trees». In: *Discrete Applied Mathematics* 315, pp. 97–103. issn: 0166-218X. doi: <https://doi.org/10.1016/j.dam.2022.03.022>. url: <https://www.sciencedirect.com/science/article/pii/S0166218X22001007>.
- Cordasco, Gennaro, Luisa Gargano e Adele A Rescigno (2022). «Pervasive Domination». In: *International Symposium on Combinatorial Optimization*. Springer, pp. 287–298.
- Cordasco, Gennaro et al. (2015). «Optimizing spread of influence in social networks via partial incentives». In: *Structural Information and Communication Complexity: 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015. Post-Proceedings 22*. Springer, pp. 119–134.
- Cordasco, Gennaro et al. (2018). «Discovering Small Target Sets in Social Networks: A Fast and Effective Algorithm». In: *Algorithmica* 80.6, pp. 1804–1833. doi: [10.1007/s00453-017-0390-5](https://doi.org/10.1007/s00453-017-0390-5). url: <https://doi.org/10.1007/s00453-017-0390-5>.
- Easley, David e Jon Kleinberg (2010). *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press.
- Leskovec, Jure e Andrej Krevl (giu. 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. <http://snap.stanford.edu/data>.
- Zhang, Junlong e Yu Luo (2017). «Degree centrality, betweenness centrality, and closeness centrality in social network». In: *2017 2nd international conference on modelling, simulation and applied mathematics (MSAM2017)*. Atlantis press, pp. 300–303.

