# Brand Protection:limitazione della diffusione di disinformazione all'interno di un social network

Matteo Calabrese
Matricola:174967

Antonio Caliò
Matricola:175056

Valerio Russo
Matricola:175057

June 7, 2016

**Abstract**

Nello studio in questione ci si è concentrati sull'implementazione di tecniche per la limitazione della disinformazione all'interno di un social network. Per modellare il problema si è immaginato di avere due processi di diffusione, che chiamaremo *campagne*, nella fattispecie ci riferiremo col nome di campagna *malevola* al processo che diffonde la disinformazione mentre col nome di campagna *benevola* indicheremo il processo di risposta, che avra il compito di limitare gli effetti della prima. Le due campagne seppure competitive tra di loro seguiranno uno stesso modello di diffusione, in particolare il modello adottato è stato *Independent Cascade*. L'obiettivo principale del lavoro è stato quello di trovare la strategia ottimale nella scelta dei nodi da cui far partire la campagna *benevola*, concentrandoci su un particolare scenario in cui la campagna *malevola* parte con un certo anticipo. Nel seguito del lavoro verranno comparati due approcci, il primo è un approccio greedy particolarmente dispendioso dal punto di vista cumputazionale contro un approccio meno oneroso basato su diverse funzioni euristiche. Dagli esperimenti é emerso che un approccio euristico, nonostante la minore complessitá, permette di ottenere risultati equiparabili all'approccio greedy.

# Contents

# 1 Introduzione

La nascita dei social network e la loro progressiva diffusione hanno portato sicuramente enormi benefici per chiunque voglia diffondere delle notizie e raggiungere velocemente una larga fetta della popolazione. Sebbene questa caratteristica possa sembrare prettamente positiva occorre guardare al rovescio della medaglia, infatti un social network può trasformarsi in una preziosissima risorsa per coloro che vogliano diffondere delle notizie non veritiere. In questo contesto si colloca il problema della *Brand Protection*, possiamo immaginare quindi l'esistenza di una compagnia,di un ente, o qualsiasi entità presente su un social network, che venga in qualche modo "attaccata" da una campagna denigratoria nei suoi confronti, la domanda che ci si pone è come reagire in seguito a tale attacco. In particolare possiamo immaginare che l'entità che abbia subito un attacco voglia rispondere ad esso attraverso la diffusione di contenuti in grado di contrastare la campagna di disinformazione messa in atto dall'attaccante, in questo scenario i fattori chiave sono essenzialmente due:

- Intercettare rapidamente la campagna diffamatoria in atto
- Rispondere all'attacco cercando di far partire un processo di diffusione che permetta di raggiungere tutti quei nodi che potrebbero essere coinvolti nella campagna *malevola*

In questo lavoro ci si concentrerà sulla seconda fase del problema, ovvero la selezione dei nodi che permettano di limitare maggiormente la campagna malevola.

# 2 Il processo di diffusione

Prima di procedere nella descrizione del problema e delle soluzioni adottate è opportuno definire il modello di diffusione delle due campagne, andando a descrivere inizialmente il modello principale, che è *Independent Cascade*, per poi passare ad un adattamento di tale processo che ne permetta l'estensione a uno scenario in cui le campagne coinvolte siano due e in competizione tra loro.

## 2.1 Independent Cascade

Independent Cascade $IC$ come descritto nella sezione 3.1 di [2], è un modello stocastico con cui si rappresenta un processo di diffusione a cascata all'interno di una rete. Secondo tale modello ogni nodo all'interno della rete può trovarsi in due stati, che sono:

- Active: un nodo che è stato coinvolto nella campagna di diffusione
- Inactive: un nodo non ancora coinvolto nella campagna di diffusione

Il cambiamento di stato può avvenire soltanto da Inactive ad Active, questo implica che quando un nodo diventa attivo non potrà più cambiare il suo stato. Il processo di attivazione avviene attraverso un fenomeno aleatorio. Quando il nodo 1 diviene attivo gli viene fornita un'unica possibilità di atti-

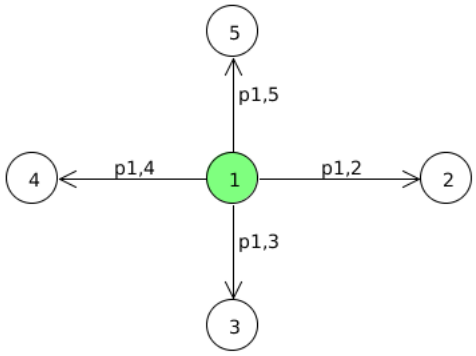



Figure 1: Processo di attivazione

vare tutti i suoi out-neighbours, quindi per ognuno degli archi uscenti verrà eseguito il tentativo di attivazione che avra successo in accordo con il peso $p_{ij}$ che rappresenta la probabilità che il nodo $i$, se attivo, riesca ad attivare il nodo $j$. Il processo continua finché non ci saranno più nodi attivabili.

## 2.2 Multi Campaign Independent Cascade

Questo modello, definito nella sezione 3.1 di [1] rappresenta un'estensione del precedente in cui le campagne in esecuzione sono due. Ci riferiremo alla campagna $C$ intesa come campagna cattiva, ovvero quella che propaga la

disinformazione, viceversa con $L$ indicheremo la campagna buona, ovvero quella di limitazione. In questo scenario ogni nodo può assumere tre stati diversi che sono:

- Inactive
- In Campaign C: attivo per la campagna C
- In Campaign L: attivo per la campagna L

Come nell'Independent Cascade una volta che un nodo risulta attivato, indipendentemente dalla campagna, non può più cambiare stato. Nello studio effettuato si assume che la probabilità $p_{ij}$, ovvero la probabilità che il nodo $n_i$ attivi il nodo $n_j$, sia uguale per entrambe le campagne. Tuttavia questa rimane un'assunzione effettuata durante lo studio, nulla impedisce che le probabilità siano diverse tra loro. Quando si parla di *MCIC* è fondamentale introdurre il concetto di tempo e dunque una priorità nel processo di attivazione che privilegi una campagna piuttosto che l'altra. Quando un nodo $n_i$ è attivo a tempo $t$ potrà eseguire un solo tentativo nell'attivare i suoi neighbour, ognuno dei quali, se attivato, cambierà stato soltanto a tempo $t+1$, inoltre il nodo $n_i$ non potrà eseguire ulteriori tentativi di attivazione. Introducendo la nozione di tempo si pone il problema di gestire le situazioni in cui un nodo $n_j$ possa essere potenzialmente attivato da più di un suo in-neighbour a favore di una campagna piuttosto che un'altra. In questo

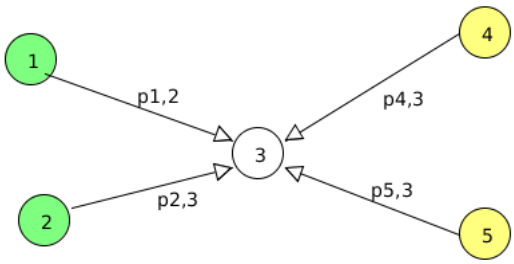



Figure 2: Nodo attivabile da entrambe le campagne a tempo t

scenario si assume che la campagna $L$ sia la prima ad eseguire il tentativo di attivazione. Se a tempo $t$ più di un nodo è in grado di eseguire il tentativo di attivazione rispetto ad una campagna, l'ordine di esecuzione dei tentativi può

essere arbitrario. Solo dopo che tutti i nodi attivi nella campagna $L$ abbiano eseguito il tentativo di attivazione, può partire allo stesso modo il processo di diffusione della campagna $C$. Il processo continua anche in questo caso finchè non ci saranno più nodi attivabili, o meglio finché tutti i nodi attivi non abbiano eseguito il proprio tentativo di attivazione.

# 3 Definizione del problema

Il problema consiste nel determinare un subset di nodi che permetta la limitazione della diffusione di una campagna malevola. Lo scenario prevede l'esistenza di due campagne $C$ e $L$ in competizione tra loro che si diffondono secondo il modello *MCIC*, un budget $k$ ed un ritardo $r$. La campagna $C$ è la prima a partire, e dopo un certo ritardo $r$ la campagna benevola $L$ inizia il processo di diffusione di contrasto. A tempo $r$ si conosce esattamente quali siano i nodi che definiremo infetti, ovvero quelli che si trovano nello stato *in campaign C*. Sulla base di tale informazione si vuole scegliere un seed di nodi inattivi, di cardinalità pari al budget $k$ dal quale far partire il processo di diffusione della campagna $L$. Nello scenario affrontato in questo studio si assume che il seed di partenza della campagna malevola sia sempre un solo nodo, ovvero quello con il maggior out degree della rete. Tale scelta, differisce dal lavoro di [1] in cui non vi sono vincoli sulle proprietà del seed iniziale al di là della cardinalità. Tale nodo, aumenta la difficoltà del problema, perchè potenzialmente potrebbe attivare un maggior numero di nodi per la campagna malevola.

## 3.1 Problem Statement

L'obiettivo è massimizzare il numero di nodi /emphsaved, ovvero quei nodi che senza l'intervento della campagna benevola $L$ risulterebbero attivi nella campagna malevola $C$.

**Definition 1.** *Dato un budget* k*, occorre determinare un* seed $A_l$ *tale che il valore di* $\Pi(A_l)$*, ossia il valore della funzione che determina la cardinalità dell'insieme dei nodi* saved *sia massimo.*

Questo vuol dire che ci si concentra principalmente sull'attivazione di quei nodi che, senza l'intervento della campagna malevola, con un'alta probabilità

sarebbero infettati dalla campagna malevola.
Lo stesso problema può essere visto da un'altra prospettiva, ovvero quella di minimizzare il numero di nodi infetti. Ciò rende lo scenario descritto differente da quanto affrontato in [2] in cui si parla di massimizzazione di influenza.

## 3.2 Esempio

Diventa subito ovvio che affinchè un nodo possa essere salvato occorre che la campagna $L$ lo raggiunga prima della campagna $C$.

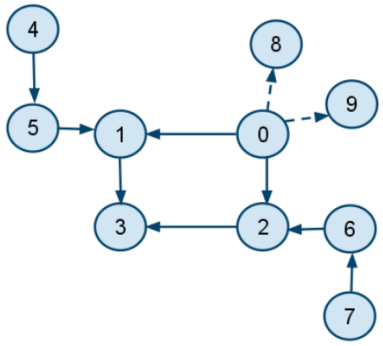



Figure 3: Esempio

L'immagine (3) ci permette di avere un'idea concreta del problema di cui stiamo parlando. Ipotizzando che il nodo 0 sia attivo per la campagna malevola ed il nostro scopo sia salvare il nodo 3, è ovvio che per ottenere tale obiettivo è necessario che questo venga raggiunto prima dalla campagna benevola.

## 3.3 Overview sul workflow

Come possiamo vedere nella figura (4) il meccanismo di diffusione vede essenzialmente tre fasi: la prima consiste nella propagazione della sola campagna malevola, questo processo dura fintanto che non scada il tempo di delay.

Esaurito il ritardo ha inizio la seconda fase: l'algoritmo decide il seed della campagna benevola e nel frattempo il processo di diffusione viene interrotto. Una volta calcolato il seed si procede con la terza fase in cui le due campagne si propagano sul network in competizione tra di loro.

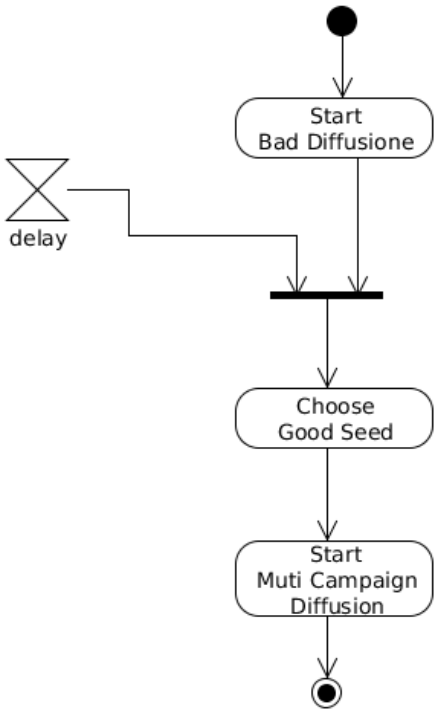



Figure 4: Workflow processo di diffusione

# 4 Soluzione

Come abbiamo già detto il problema consiste nella selezione dei nodi da cui far partire la campagna di contrasto a quella malevola. Nella costruzione di tale seed si sono adottati diversi approcci, che vedremo più in dettaglio nel seguito. L'obiettivo dello studio è stato confrontare le prestazioni delle diverse soluzioni implementate. In particolare si vogliono confrontare i risultati ottenuti tramite un approccio greedy basato su "metodo Montecarlo", quindi molto oneroso dal punto di vista computazionale, contro altre soluzioni

basate su diverse funzioni euristiche per la selezione dei nodi come trattato nella sezione 4.3 in [1].

## 4.1 Approccio Greedy

L'approccio consiste nel selezionare quei nodi che presentano il maggior contributo marginale, o meglio quei nodi che permettano di incrementare maggiormente l'insieme dei nodi salvati. Il contributo marginale di un nodo si calcola come:

**Definition 1.** *Sia* f *la funzione che dato un insieme* S *in ingresso restituisca il numero di nodi salvati,il contributo marginale di un nodo* n *sarà dato da* $f(S \cup n) - f(S)$*, ovvero quei nodi che l'insieme* S *non riesce a salvare ma che vengono salvati dall'aggiunta di* n *allo stesso insieme.*

Essendo il processo di diffusione di natura stocastica, per avere una stima plausibile del contributo marginale occorre eseguire il calcolo della funzione $f$ un numero elevato di volte, nel lavoro presentato per il calcolo si sono utilizzati un numero di run pari a 1000.

```
Data: G: grafo, L: nodi infetti, k: budget, start: indice di partenza
      diffusione della campagna C
Result: seed campagna L
maxGain = -maxInteger;
alreadySaved = 0;
SEED = emptyList;
while i < k do
    for x ∈ G do
        if x not in L and x not in SEED then
            append x to SEED;
            marginalGain = marginalGain
             (L,SEED,start,alreadySaved);
            if marginalGain > maxGain then
                candidate = x;
                maxGain = marginalGain;
            end
            remove x from SEED;
        end
    end
    append candidate to SEED;
    alreadySaved += maxGain;
    maxGain = -maxInteger;
    i+=1;
end
return SEED;
```

Il metodo marginalGain non fa altro che eseguire il processo di diffusione e restituisce il valore del guadagno marginale. [1]

## 4.2 Approccio Euristico

Nello studio sono state prese come riferimento le tre diverse funzioni euristiche che vedremo in dettaglio nel seguito. Tali implementazioni non fanno riferimento a quanto proposto nell'articolo [1] bensì sono frutto di una nostra interpretazione.

[1] Il "metodo Montecarlo" richiederebbe un numero di simulazioni maggiore di quello utilizzato, purtroppo non è stato possibile per via della mancanza delle risorse di calcolo necessarie.

**Out Degree** Questa funzione euristica prende come seed i nodi col maggior out degree tra tutti quelli che non sono ancora stati infettati dalla campagna malevola. In altre parole non si fa altro che prendere i $k$ nodi più influenti della rete non ancora infetti.

**Data:** G: grafo, L: nodi infetti, k: budget, start: indice di partenza diffusione della campagna C
**Result:** seed campagna L
OD = emptyList;
**for** $x \in G$ **do**
  **if** *x is not infected* **then**
    append x, outDegree(x) to OD;
  **end**
**end**
**return** first k node from reverseOrder(OD);

Il metodo prima crea una lista (nodo,out degree del nodo) dopodiché restituisce i primi k nodi ottenuti ordinando gli elementi sulla base del valore out degree decrescente.

**Early Infected** Questa funzione euristica si pone come obiettivo quello di selezionare i $k$ nodi che hanno la maggiore probabilità di venire infettati nella prima fase della propagazione, o meglio a time step $r$. Anche in questo caso per ottenere la probabilità di attivazione di un nodo occorre simulare un numero elevato di volte il processo di infezione. Attraverso l'utilizzo di questa funzione si andranno a selezionare quei nodi che sono direttamente connessi ai nodi gia infetti.

```
Data: G: grafo, L: nodi infetti, k: budget, start: indice di partenza
      diffusione della campagna C
Result: seed campagna L
EI = emptyDict (node:activationProbability) ;
for x ∈ L do
    xDict = out neighbour di x;
    for key ∈ xDict do
        if key is not infected and key is not in EI then
            INWEIGHT = peso di tutti gli archi entranti in key che
              partono da nodi infetti;
            activationProbability =
              getActivationProbability(key,INWEIGHT);
            EI[key]=activationProbability;
        end
    end
    return first k nodes from reverseOrder(EI);
end
return first k node from reverseOrder(OD);
```

Il metodo getActivationProbability prende in ingresso un nodo e la lista degli archi entranti ed esegue il tentativo di attivazione. Quindi anzichè simulare l'intero processo di diffusione ci si concentra soltanto su quei nodi che sono in prossimità dei nodi infetti.

**Largest Infectees** Questa euristica è molto simile alla precedente, con l'eccezione che piuttosto di prendere i nodi che saranno infetti con un'alta probabilità nella prima fase della propagazione, si scelgono quei nodi che se infettati propagherebbero maggiormente l'infezione. Il calcolo di questa funzione è abbastanza oneroso, infatti consiste nel simulare il processo di diffusione della campagna malevola, una volta spirato il delay $r$,un elevato numero di volte, e ad ogni nodo viene associato un valore $v_i$, inizialmente 0, che viene incrementato ogni qual volta il nodo, indipendentemente che sia infetto o meno, si trovi sullo shortest path che congiunge un nodo infetto ad un altro infetto dopo il delay $r$.

```
Data: G: grafo, L: nodi infetti, k: budget, start: indice di partenza
      diffusione della campagna C
Result: seed campagna L
TORESTORE = reduceGraph(G,L);
SPDICT = createShortestPath(L);
LI = emptyDict;
count = 0;
while count < run do
    SL = SimulateIC(G,L,start);
    for x ∈ SL do
        if x in SPDICT then
            for yinSPDICT[x] do
                LI[y] += 1;
            end
        end
        count += 1;
    end
end
restoreGraph(TORESTORE);
return first k nodes from reverseOrder(LI);
```

Per ragioni di efficienza nell'implementazione proposta si riduce inizialmente la dimensione del grafo, eliminando quegli archi che congiungono due nodi infetti. Inoltre tramite la funzione createShortestPath si creano a priori i path che congiungono i nodi infetti a quelli ancora inattivi.

# 5 Implementazione

## 5.1 Ambiente di sviluppo

Per sviluppare l'applicazione si è scelto di utilizzare il linguaggio di programmazione *Python* utilizzando prevalentemente nella sua accezione di linguaggio Object Oriented. Per la lettura e la manipolazione del grafo ci si è appoggiati invece alla libreria python *networkx*.

## 5.2 Diagramma della classi

La figura (5) rappresenta il diagramma delle classi dell'applicazione. Gli elementi principali sono:

- **GraphHandler** è la classe dedicata alla gestione del grafo
- **RealGraphHandler** estende la classe GraphHandler, durante l'esecuzione accede e modifica direttamente le strutture dati del grafo
- **SimulationGraphHandler** è la classe utilizzata quando si devono simulare i processi di diffusione, anziché modificare le strutture del grafo mantiene le modifiche in memoria. Questo ci permette di non dover eseguire il rollback dei processi simulati.
- **Heuristic** è l'interfaccia comune a tutte le euristiche, per poter eseguire necessita di un riferimento ad un oggetto GraphHandler.

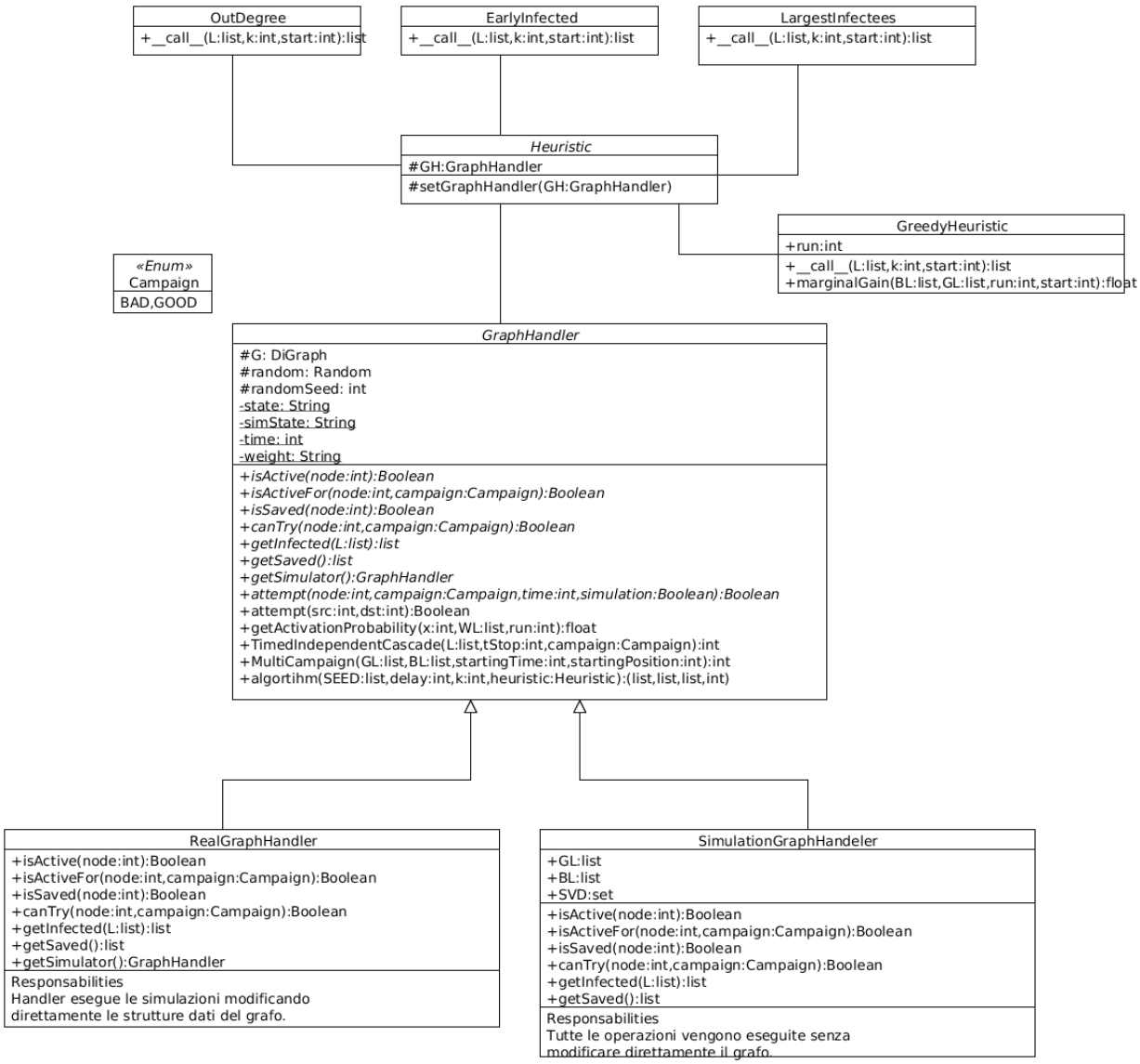



Figure 5: Diagramma delle classi

## 5.3 Esecuzione dell'applicazione

Per eseguire l'applicazione occorre eseguire lo script *start.sh* specificando i parametri nella tabella (1):

| Opzione | Significato | Default |
|---|---|---|
| -g | Path del file contenente il grafo | Obbligatorio |
| -w | [T,F] Indica se il grafo é pesato o meno | T(true) |
| -h | [E,O,G,L] Indica l'euristica da utilizzare | O(outdegree) |
| -d | Indica il delay della campagna benevola | Obbligatorio |
| -s | Indica la numerosita del seed della campagna malevola | Obbligatorio |
| -k | Indica il budget a disposizione | Obbligatorio |
| -r | Indica il numero di run da eseguire | 100 |
| -rs | Indica il seed del generatore random | None |

Table 1: Parametri di esecuzione

Una volta specificati i parametri in ingresso verrà istanziato un GraphHandler e la funzione euristica specificata. L'applicazione eseguirà seguendo il workflow specificato nella figura (4). Alla fine dell'esecuzione verranno forniti all'utente i valori:

- Numero di nodi salvati
- Numero di nodi attivi nella campagna malevola
- Numero di nodi attivi nella campagna benevola
- Tempo di esecuzione

Il tempo di esecuzione rappresenta il numero di passi compiuti dall'algoritmo prima che il processo di diffusione sia terminato per entrambe le campagne, ovvero quando per ambedue le campagne si sia raggiunta la condizione per cui non ci siano più nodi attivabili.

## 5.4 La nozione di tempo

Nella sezione 2.1 abbiamo già affrontato e descritto il modello di diffusione *Independent Cascade*, questo è un processo stocastico in cui non vi è una chiara rappresentazione del tempo. Tuttavia nell'ambito descritto dal nostro problema il tempo gioca un ruolo fondamentale, in quanto affinché un nodo possa essere salvato la campagna benevola deve raggiungerlo prima della sua avversaria. Il tempo è stato rappresentato sotto forma di time step. Si parte con $t = 0$ e l'avanzamento a $t + 1$ avviene solo quando ogni nodo attivo a tempo $t$ abbia eseguito il tentativo di propagazione verso tutti i suoi nodi

vicini, dando precedenza nell'esecuzione del tentativo ai nodi attivi nella campagna benevola.

# 6 Fase di test

Nella fase di test ci si è concentrati principalmente nella determinazione di un approccio ottimale alla soluzione del problema, per questo motivo durante le simulazioni si cercherà di mettere in evidenza le principali caratteristiche di ogni algoritmo proposto.

## 6.1 Dataset Utilizzati

Le simulazioni sono state effettuate su tre dataset differenti

1. Due grafi generati con modello Baràbasi[2], descritto in
   (a) barabasi3k
      - Numero di nodi: 100
      - Numero di archi: 3725
      - Degree average: 37.25
      - Influence average: 0.507

   (b) barabasi95k
      - Numero di nodi: 1000
      - Numero di archi: 94950
      - Degree average: 94.95
      - Influence average: 0.507

   I parametri utilizzati per la creazione di tali grafi sono stati:

   - $p = 1$ che indica il coefficiente della power law. Con questo settaggio si genera un grafo con preferential attachment lineare.

[2]Si fa riferimento al modello Barabàsi-Albert (BA) che è un algoritmo utilizzato per generare delle reti sociali in maniera randomatica. Per la generazione è stato utilizzato il software "R" con la libreria iGraph

- $m = 50$ per il primo grafo ed $m = 100$ per il secondo grafo. Questo parametro indica il numero di tentativi di creazione di un arco eseguiti per ogni step. Naturalmente il grado di connessione nel secondo esempio sarà maggiore.

Per quanto riguarda invece la generazione dei pesi sugli archi,questi vengono generati con valore inversamente proporzionale all' in degree del no

2. Un dataset rappresentante la componente gigante del network di *Instagram*, con le caratteristiche:

   - Numero di nodi: 17521
   - Numero di archi:617499
   - Degree average: 35.243
   - Influence average: 0.985

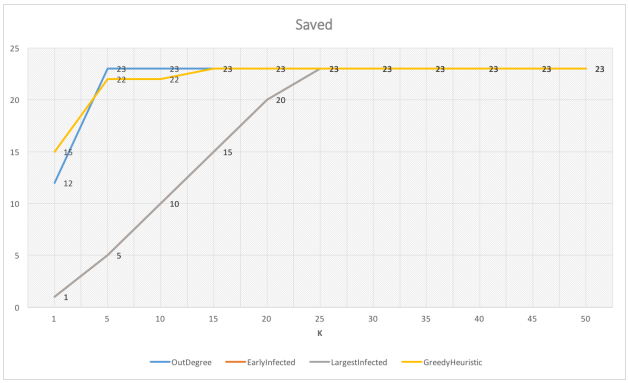
Con il termine *'Influence average'* si indica la probabilità media che un nodo propaghi la campagna ad un suo out-neighbour, quindi nelle simulazioni abbiamo assunto che le campagne si propaghino con la stessa velocità.

## 6.2 Settings

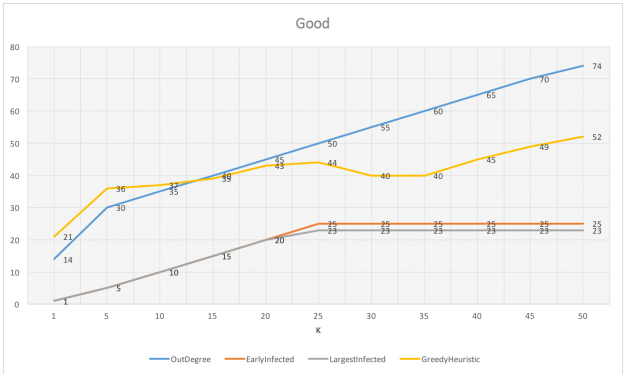
Le simulazioni sono state condotte sotto le ipotesi che la campagna malevola parta sempre con un time step di vantaggio e il processo di propagazione abbia inizio sempre a partire da un unico nodo, scelto in modo tale che sia quello col maggior out degree. Nelle varie simulazioni quindi si è mantenuto fisso il ritardo mentre il valore del budget $k$ si è fatto variare con valori sempre crescenti.

## 6.3 Risultati

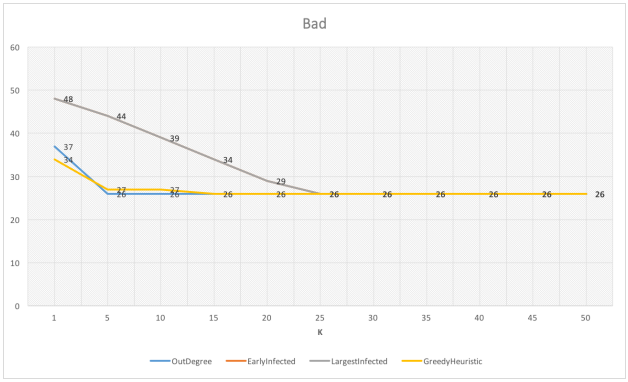
I grafici sono raggruppati sulla base del grafo su cui è stata effettuata la simulazione, in particolare per ogni istanza di simulazione verrà mostrato in ordine l'andamento dei nodi *saved, good, bad*.




(a) Saved

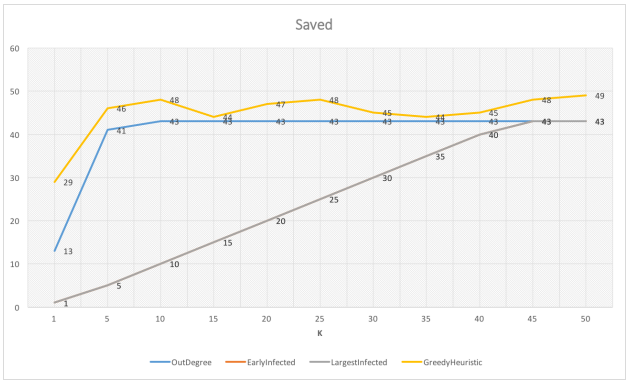



(b) Good

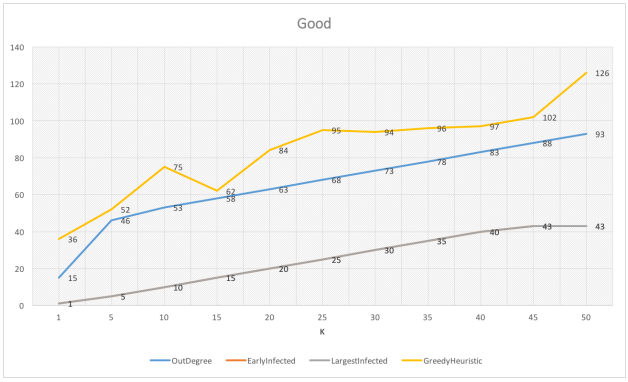



(c) Bad

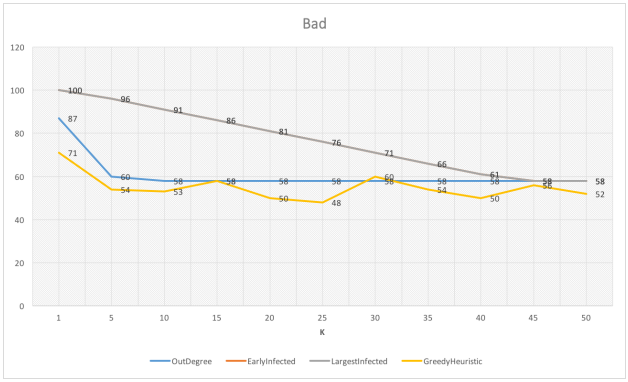
Figure 6: Grafici barabasi3k




(a) Saved

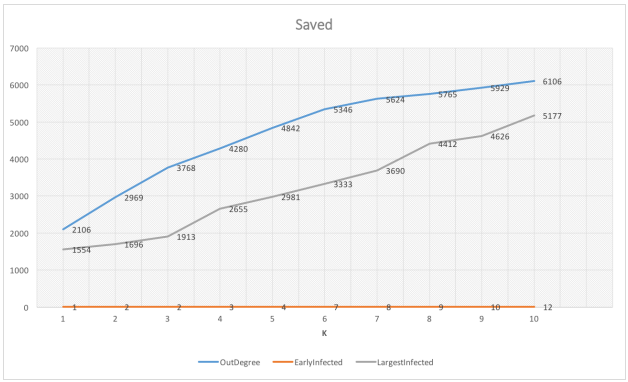



(b) Good

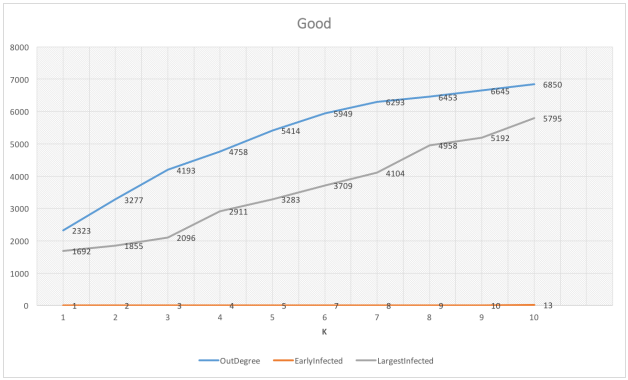



(c) Bad

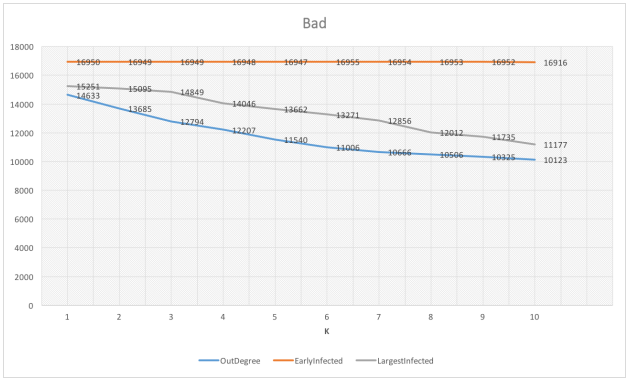
Figure 7: Grafici barabasi95k




(a) Saved




(b) Good




(c) Bad

Figure 8: Grafici instagramLCC

Nella figure (6) e (7) dei *saved* e *bad* si evince chiaramente la convergenza delle euristiche al crescere del budget. Un risultato non atteso è però l'andamento delle due euristiche Early Infeceted e Largest Infectees, infatti da quanto riportato nella sezione 5 di [1] la seconda funzione avrebbe dovuto garantire i risultati migliori. Nel nostro caso invece tale comportamento è praticamente inverso, infatti coincide esattamente con la Early Infected, ovvero la funzione che prometteva di essere la peggiore. La motivazione di tale andamento risiede nella topologia del grafo utilizzato. Nella fattispecie l'elevato grado di connessione dei grafi barabasi3k e barabasi95k fa si che lo shortest path medio tra le coppie di nodi sia pari rispettivamente a 2.01 e 2.55. In questo scenario la funzione Largest Infectees, nel cui calcolo è coinvolta la determinazione degli shortest path che partono dal set di nodi già infetti, si comporta esattamente come la Early Infected, in quanto verranno scelti quei nodi che si trovano in prossimità della componente infetta. Per quanto riguarda la figura (8) possiamo osservare ancora una volta la dominanza della funzione Out Degree. Anche in questo caso i risultati attesi sulla base di quanto visto nella sezione 5 di [1] sono leggermente differenti. La funzione Largest Infectees avrebbe dovuto manifestare un seppur leggero vantaggio nei confronti della Out Degree. Ancora una volta la motivazione risiede nella forte connessione del grafo, infatti lo shortest path medio è prossimo al barabasi3k.

## 6.4 Analisi dei risultati

I risultati dimostrano che col crescere del budget, tutte gli algoritmi permettono di ottenere le stesse prestazioni in termini di nodi salvati. Tuttavia se valutassimo gli algoritmi anche da un punto di vista delle risorse computazionali richieste, la scelta della migliore strategia ricadrebbe sulla prima euristica, ovvero *Out Degree*

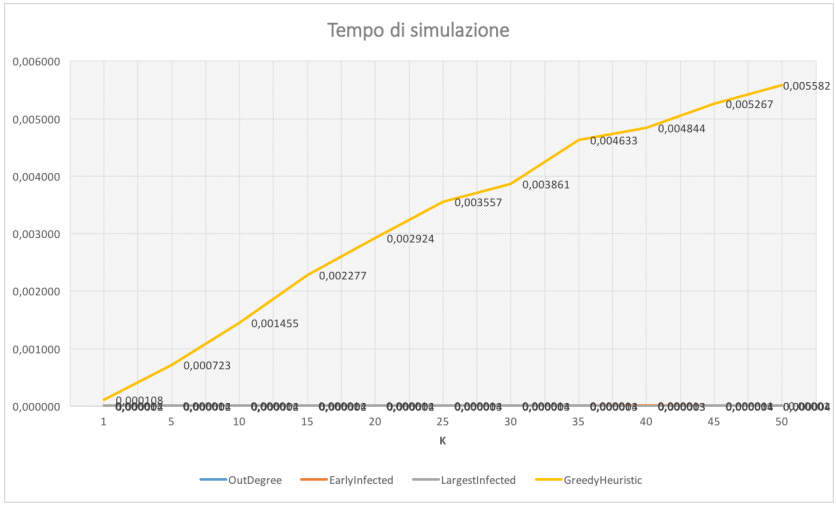
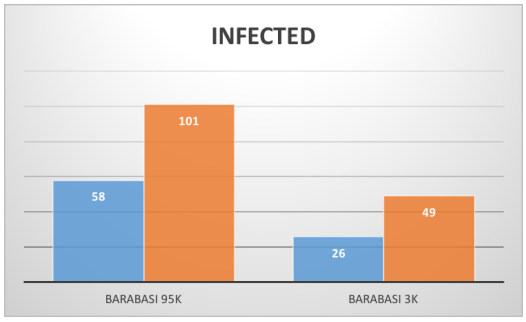


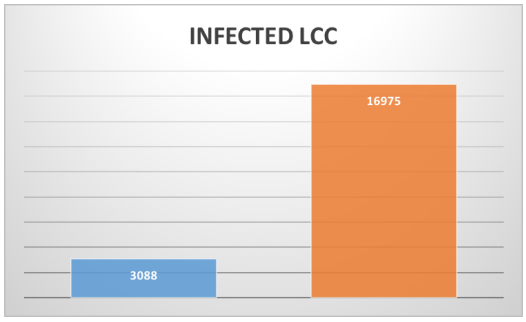

Figure 9: Tempo di simulazione barabasi3k

Il grafico (9) mostra il tempo in secondi richiesto dall'esecuzione dei vari algoritmi. Possiamo notare infatti che l'algoritmo greedy richiede una maggiore quantità di risorse computazionali,tale differenza si manifesta anche con i grafici di modeste dimensioni,nonostante le prestazioni siano in alcuni casi persino peggiori della funzione *Out Degree*, che di contro richiede un tempo di esecuzione minore. Il perchè del comportamento dell'algoritmo greedy risiede nella struttura del grafo, infatti il contributo marginale tende a decrescere rapidamente già dopo la selezione dei primi nodi. Questo risultato è dovuto al fatto che la capacità di propagazione della campagna malevola, in assenza della di quella di contrasto, non le permette di attivare tutti i nodi del grafo.

Nell'immagine (10) possiamo osservare in blu il numero di nodi che risultano infetti allo scadere del delay, mentre in arancione osserviamo il numero di nodi che risulterebbero infetti se non intervenisse la campagna di contrasto. Nella figura 10.a possiamo notare come già allo scadere del delay, la campagna malevola riesca ad infettare praticamente la metà dei nodi potenzialmente infettabili, i quali risultano essere il 10% e il 50% della totale popolazione, rispettivamente per il grafo barabasi95k e barabasi3k. Discorso differente invece per quanto riguarda il grafo di instagramLCC, riportato nella figura 9.b, infatti in questo caso se non intervenissimo con la campagna benevola, il seed di partenza di quella malevola è praticamente in grado di infettare la totalità della rete. Questo comportamento ci permette di analizzare i risultati fin qui visti sotto un'altra ottica, infatti se ci soffermiamo sul numero di




(a) barabasi




(b) instagramLCC

Figure 10: Numero di nodi infetti

nodi infetti nelle figure (6),(7) e (8) notiamo immediatamente che tale valore tende, all'aumentare di k, ad una precisa quantità, ovvero al numero di nodi gia infetti prima dell'intervento della campagna benevola. Questa potrebbe essere considerata come una condizione di idealità, infatti raggiungerla significherebbe aver contrastato perfettamente la campagna malevola. Osservando i risultati sotto questa nuova luce e valutando l'andamento del numero di nodi infetti nella rete, è interessante determinare quale strategia riesca a garantire prima questa condizione. Possiamo notare allora che l'algoritmo greedy e la funzione euristica out degree permettono di raggiungere limitare interamente la campagna malevola con un k piu piccolo rispetto alle altre due strategie. In altre parole ci permettono di fermare la campagna malevola spendendo il minor budget possibile.

# 7 Conclusioni

L'analisi condotta ha fatto emergere che l'approccio migliore è sicuramente quello che utilizza come funzione euristica la *Out Degree*. Questo risultato in antitesi con quanto rilevato in [1] ci ha lasciato dapprima stupiti,ma approfondendo la questione siamo riusciti a trovare la motivazione, che risiede nella forte dipendenza dei risultati dal grafo su cui si conducono gli esperimenti. Quindi possiamo affermare che nell'affrontare il problema non esista una soluzione che sia univocamente migliore delle altre, ma che la scelta della migliore strategia debba passare obbligatoriamente da un'analisi approfondita della topologia del grafo che si sta utilizzando. I parametri più interessanti da valutare sono il grado di connessione del grafo e la capacità propagativa della campagna malevola. Un altro risultato emerso dagli esperimenti è inoltre l'importanza della tempestività dell'intervento, infatti sebbene siano stati omessi i grafici in questo lavoro, come ci si aspettava le prestazioni di tutti gli algoritmi, al crescere del ritardo della campagna benevola, degradano velocemente, non permettendo cosi di salvare nessun nodo.

# References

[1] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th*

*International Conference on World Wide Web*, WWW '11, pages 665–674, New York, NY, USA, 2011. ACM.

[2] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM.