

# **Analisi delle reti sociali**

## *Majority Domination e Cascade nei Grafi*

Riccardo Martiniello

Marco Cappiello

**Sommario**

Analisi di reti sociali ..... 3

La rete ..... 4

Funzioni scelte ..... 7

    I costi ..... 7

    Funzioni per la scelta del Seed Set ..... 8

Funzione Influenza ..... 11

Sperimentazione ..... 12

Conclusione ..... 14

# Analisi di reti sociali

Una rete sociale è una struttura composta da individui o entità e le relazioni tra essi. Lo studio di una rete sociale permette di ottenere informazioni sulle dinamiche sociali e le interazioni tra individui, come la diffusione di un'idea, di un prodotto o di un'epidemia.

Graficamente vengono rappresentate come grafi composti da nodi ed archi.

Alla base dello studio una rete sociale c'è il concetto di Dominating Set, ovvero un sottoinsieme di nodi  $D$  appartenente a un grafo  $G$  tale che:

$$\forall v \in V-D \text{ vale che } |d(v) \cap D| \geq 1$$

Un'estensione di questo concetto è il Majority Dominating Set:

$$\forall v \in V-D \text{ vale che } |d(v) \cap D| \geq \frac{d(v)}{2}$$

Un aspetto interessante delle reti sociali è il poter influenzare i nodi della rete partendo da un piccolo set iniziale, a tal proposito introduciamo il processo di Majority Dynamical Process of Influence Diffusion.

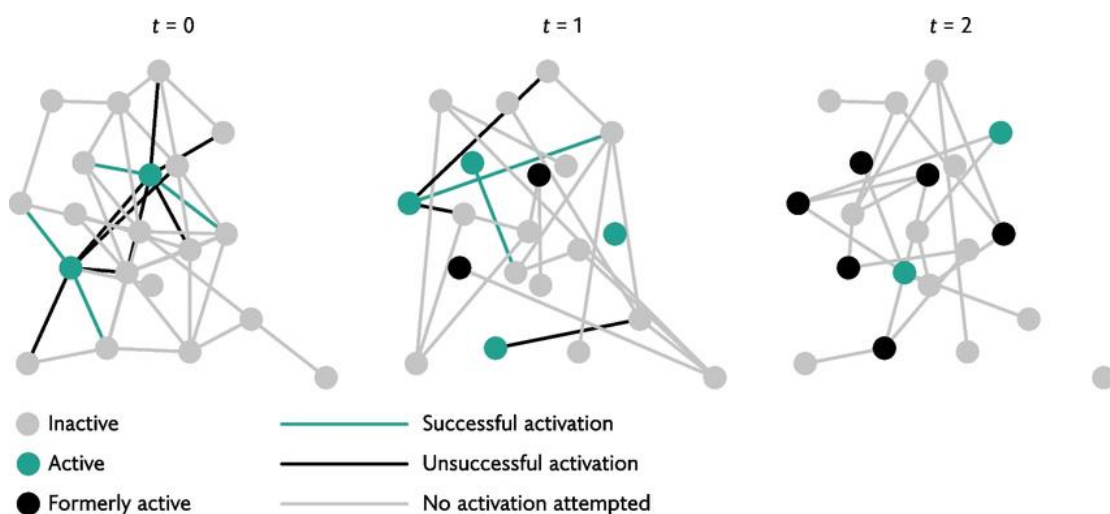


Figura 1 Esempio di Influence Diffusion

Questo processo ci permette, partendo da un Seed set iniziale, di trovare quali nodi verranno influenzati in modo diretto (attraverso i nodi confinanti) e in modo indiretto (attraverso i nodi influenzati e i loro nodi confinanti) dal set iniziale.

Possiamo ampliare questo concetto considerando i singoli costi assegnati ad ogni nodo della rete selezionando quindi i nodi che rientrano in un budget specificato.

Purtroppo, scegliere il set iniziale è un problema difficile che si può risolvere soltanto prendendo in considerazione ogni singola configurazione di partenza per trovarne una ottimale, e questo lo fa rientrare nei problemi NP-HARD.

Fortunatamente possiamo sfruttare delle euristiche in grado di trovare una soluzione non necessariamente ottimale a questo problema.

In questo documento verranno presentati tre euristiche per scegliere il Seed Set iniziale su una rete considerando tre possibili configurazioni dei costi per ogni nodo.

Nelle sezioni successive, verrà fornita una descrizione dettagliata della rete sociale scelta, seguita da una descrizione delle funzioni utilizzate, dei costi assegnati e delle sperimentazioni effettuate.

## La rete

La rete utilizzata per il progetto è la rete "ca-netscience", che è un grafo non diretto e non pesato che rappresenta le collaborazioni tra ricercatori. Questa rete ha caratteristiche strutturali che possono essere molto interessanti per studiare fenomeni di diffusione e cascata, come il modello cost majority cascade. La sua struttura complessa e le connessioni tra i nodi possono simulare scenari reali di diffusione dell'informazione.

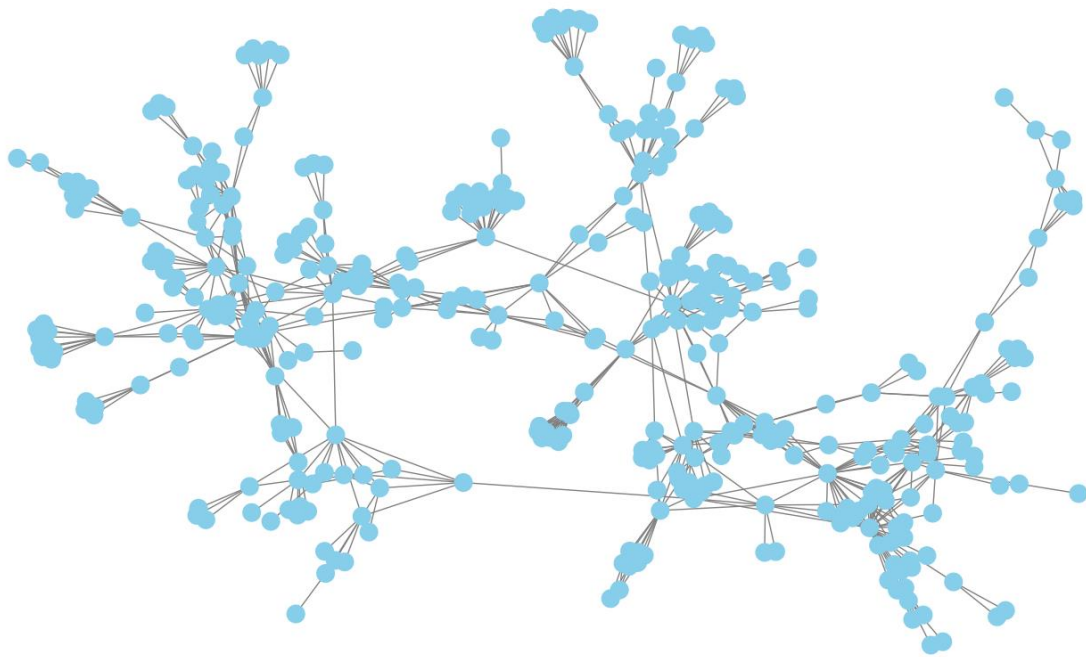
I nodi rappresentano gli scienziati e gli archi rappresentano le collaborazioni tra di loro. Questa eterogeneità nei nodi (dove alcuni scienziati sono molto più collaborativi di altri) è perfetta per testare algoritmi di selezione dei seed set.

La rete è ben documentata e disponibile pubblicamente ed è composta da 379 nodi e 914 collegamenti tra di essi, la documentazione riporta anche il grado massimo presente nella rete (34) e il grado medio (4).

Usando un semplice script python è possibile stampare visivamente la rete per poter osservare i cluster presenti.

```
pos = nx.spring_layout(G) |
node_color = []
for node in G.nodes():
    if red_nodes and node in red_nodes:
        node_color.append('red')
    elif purple_nodes and node in purple_nodes:
        node_color.append('purple')
    else:
        node_color.append('skyblue')
nx.draw(G, pos, with_labels=False, node_color=node_color, edge_color='gray', node_size=200, font_size=20)
plt.show()
```

*Figura 2 Script python che stampa il grafo relativo alla rete*



*Figura 3 La rete ca-netscience*

Dalla Figura 2 possiamo osservare la presenza di numerosi cluster, che sono gruppi di nodi densamente connessi tra loro. Questi cluster possono ostacolare il processo di diffusione delle influenze, soprattutto se la loro densità è maggiore rispetto ai cluster confinanti già influenzati. Inoltre, molti cluster sono collegati da nodi ponte (bridge), che fungono da collegamenti cruciali tra diversi cluster. La presenza di questi nodi ponte è fondamentale perché, se non influenzati, possono bloccare completamente la diffusione dell'influenza tra i cluster.

In genere, questo problema potrebbe essere superato influenzando un numero sufficiente di nodi all'interno di un cluster per diminuirne la solidità e facilitare la diffusione dell'influenza. Tuttavia, nel nostro caso, la scelta dei nodi deve rispettare un budget limitato, il che rende più difficile

garantire questa soluzione. Quindi, i nodi ponte e i cluster densi rappresentano sfide significative nel contesto del modello cost majority cascade che stiamo studiando.

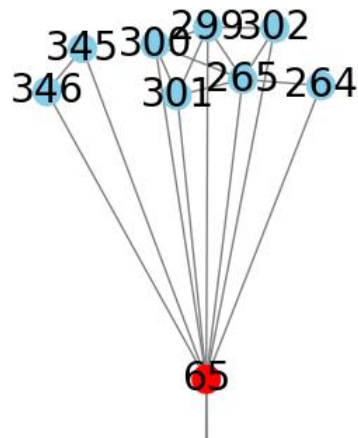


Figura 4 Esempio aggiungendo il 65 al seed set

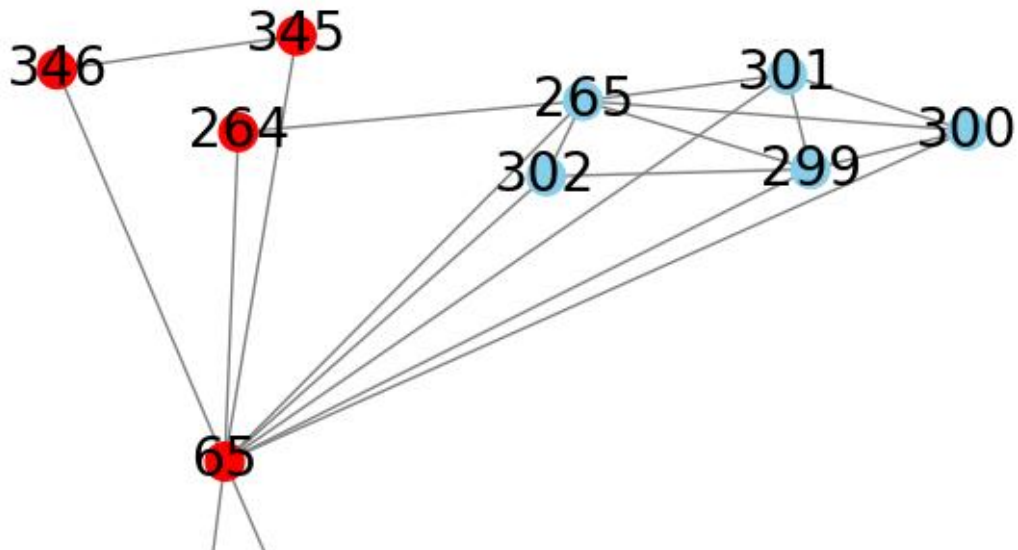


Figura 5 il grafo dopo il processo di cascade

Prendendo in considerazione una porzione specifica del grafo possiamo osservare per esempio il nodo 65 che collega il resto della rete con i cluster formati dai nodi {265, 301, 300, 299, 302}.

Considerando il nodo 265 che appartiene ad un cluster di densità 4/6 possiamo notare che questo non potrà mai essere influenzato dai nodi 65 e 264.

## Funzioni scelte

### I costi

Ogni nodo ha un costo assegnato, generalmente questi costi dipendono dal tipo di rete analizzata, per i fini di questo progetto però abbiamo scelto di utilizzare tre tipologie di funzioni costi  $c: V \rightarrow \mathbb{N}$  diverse per il calcolo del costo dei singoli nodi che compongono la rete:

- 1) **Funzione costi random:** Ad ogni nodo viene associato un valore intero casuale in un determinato range (nel nostro caso tra 5 e 10). La sua definizione viene mostrata attraverso questo semplice script python

```
random.seed(42)
costs = {node: random.randint(a=5, b=10) for node in Graph_nx.nodes()}
```

Figura 5 script Python che definisce la funzione random

- 2) **Funzione Costi Grado del Nodo/2:** Ad ogni nodo viene associato un costo pari a  $\frac{d(u)}{2}$  dove  $d(u)$  è il grado del nodo  $u$ . Questo è stato implementato come segue:

```
costs = {node: Graph_nx.degree(node)/2 for node in Graph_nx.nodes()}
```

Figura 6 script Python che definisce la funzione costi grado del nodo/2

- 3) **Funzione Costi basata sulla Centralità del Vicinato:** Ad ogni nodo viene associato un valore basato sulla sua centralità del vicinato. Questo valore è calcolato come  $\frac{1}{closeness\_centrality[node]}$  se la centralità del nodo non è zero, altrimenti viene impostato a Infinito. Questo è lo script Python che rappresenta tale funzione:

```
closeness_centrality = nx.closeness_centrality(Graph_nx)
costs = {node: 1 / closeness_centrality[node] if closeness_centrality[node] != 0 else float('inf') for node in Graph_nx.nodes()}
```

Figura 7 script Python che definisce la funzione costi basata sulla Centralità del Vicinato

## Funzioni per la scelta del Seed Set

I nodi scelti per far partire la cascade vengono selezionati da uno dei tre algoritmi presentati utilizzati.

1. Il primo algoritmo che utilizziamo è chiamato **Cost-Seeds-Greedy**:

Di seguito riportiamo lo pseudocodice dell'algoritmo

Algorithm	Cost-Seeds-Greedy $(G, k, c, f_i)$
1	$S_p = S_d = \emptyset.$
2	<b>repeat</b>
3	select $u = \arg \max_{v \in V - S_d} \frac{\Delta_v f_i(S_d)}{c(u)}$
4	$S_p = S_d$
5	$S_d = S_p \cup \{u\}$
6	<b>until</b> $c(S_d) > k$
7	<b>return</b> $S_p$

L'algoritmo "Cost-Seeds-Greedy" seleziona un insieme di nodi (seed set) che massimizza l'influenza complessiva nel grafo, rispettando un vincolo di costo totale. A ogni iterazione, aggiunge il nodo con il miglior rapporto tra incremento di influenza e costo fino a esaurire il budget. Quindi, viene selezionato un insieme di nodi da un grafo che massimizza una funzione obiettivo  $f_1(S)$ :

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

Questa funzione serve a quantificare l'incremento dell'influenza ottenibile aggiungendo un nodo al set S di nodi influenzati, tenendo conto sia del numero di vicini già influenzati sia del grado del nodo. L'algoritmo, quindi, sceglie i nodi da aggiungere a S in modo da massimizzare questo valore di copertura rispetto al costo dei nodi stessi.

Di seguito la nostra implementazione in python:



```
def cost_seeds_greedy(graph, k, costs):
    setGrafo=set(graph.nodes())
    Sp = set()
    Sd = set()
    print(sorted(setGrafo, key=lambda v: delta_v-fi(graph, Sd, v) / costs[v]))
    budget_used = 0
    while True:
        try:
            u = max(setGrafo - Sd, key=lambda v: delta_v-fi(graph, Sd, v) / costs[v])
            if delta_v-fi(graph, Sd, u)<=0:
                break

            if sum(costs[v] for v in Sd) + costs[u] <= k:
                Sp = Sd
                Sd = Sp.union({u})
                budget_used += costs[u]
                print("Budget utilizzato:", budget_used)
                if budget_used==k:
                    break
                print(Sd)
            else:
                discard=set()
                discard.add(u)
                setGrafo=setGrafo-discard
        except:
            break
    return Sd
```

```
def delta_v-fi(graph, S, u):
    S_with_u = S | {u}
    total_with_u = 0
    for v in graph.nodes():
        min_neighbors_in_S_with_u = min(len(set(graph.neighbors(v)) & S_with_u), math.ceil(graph.degree(v) / 2))
        total_with_u += min_neighbors_in_S_with_u
    total = 0
    for v in graph.nodes():
        min_neighbors_in_S = min(len(set(graph.neighbors(v)) & S), math.ceil(graph.degree(v) / 2))
        total += min_neighbors_in_S
    return total_with_u - total
```

2. Il secondo algoritmo che utilizziamo è una variante del primo algoritmo, **Cost-Seeds-Greedy**, nel senso che allo stesso algoritmo viene applicata una funzione diversa  $f_2(S)$ :

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

La funzione  $f_2(S)$  valuta l'influenza di un set di nodi  $S$  nel grafo considerando il grado dei nodi e il numero di vicini influenzati. Per ogni nodo  $v$  e i suoi vicini influenzati  $S$ , calcola un valore che riflette quanto ogni nodo è influenzato dal seed set. Utilizza il grado di ciascun nodo per ponderare l'influenza ricevuta.

Di seguito la nostra implementazione in python:

```

def cost_seeds_greedy_f2(graph, k, costs):
    setGrafo = set(graph.nodes())
    Sp = set()
    Sd = set()

    print(sorted(setGrafo, key=lambda v: delta_v_fi_f2(graph, Sd, v) / costs[v]))
    budget_used = 0
    while True:
        try:
            u = max(setGrafo - Sd, key=lambda v: delta_v_fi_f2(graph, Sd, v) / costs[v])
            if sum(costs[v] for v in Sd) + costs[u] <= k:
                Sp = Sd
                Sd = Sp.union({u})
                budget_used += costs[u]
                print("Budget utilizzato:", budget_used)
                if budget_used == k:
                    break
                print(Sd)
            else:
                discard = set()
                discard.add(u)
                setGrafo = setGrafo - discard
        except:
            break
    return Sd

```

```

def f2(graph, S):
    total_influence = 0
    for v in graph.nodes():
        num_neighbors_in_S = len(set(graph.neighbors(v)) & S)
        dv = math.ceil(graph.degree(v) / 2)
        influence = sum(max(dv - i + 1, 0) for i in range(1, num_neighbors_in_S + 1))
        total_influence += influence
    return total_influence

```

2 usages

```

def delta_v_fi_f2(graph, S, u):
    S_with_u = S | {u}
    return f2(graph, S_with_u) - f2(graph, S)

```

3. Il terzo algoritmo chiamato **MySeeds** seleziona un insieme di nodi (seed set) basato sulla centralità di betweenness dei nodi in un grafo. Quindi, questo algoritmo parte col calcolare la centralità di betweenness per ogni nodo. Dopodiché ordina i nodi in base alla loro centralità di betweenness in ordine decrescente. Itera sui nodi ordinati aggiungendoli al

seed set finché il budget disponibile lo permette e infine restituisce il seed set che massimizza la betweenness centrality senza superare il budget.

Di seguito la nostra implementazione python dell'algoritmo:


```
def my_seeds_betweenness(graph, k, costs):  
    S = set()  
    budget_used = 0  
    betweenness = nx.betweenness centrality(graph)  
    nodes_sorted_by_betweenness = sorted(betweenness, key=betweenness.get, reverse=True)  
    for node in nodes_sorted_by_betweenness:  
        if budget_used + costs[node] > k:  
            break  
        S.add(node)  
        budget_used += costs[node]  
    print("Budget utilizzato:", budget_used)  
    print(S)  
    return S
```

## Funzione Influenza

Una volta utilizzata una delle funzioni descritte precedentemente, dobbiamo simulare il cascade per conoscere i nodi della rete che verranno influenzati partendo dal nostro set calcolato.

L'idea alla base della funzione di influenza è quella di prendere inizialmente in considerazione i nodi del seed set come nodi influenzati, successivamente vengono influenzati tutti i nodi che hanno almeno metà dei loro adiacenti influenzati. I nodi influenzati potranno quindi potenzialmente influenzare i loro vicini fino a quando non verranno a scontrarsi con un cluster di densità  $> 1/2$ .

Di seguito riportiamo la nostra implementazione:

```
def majority_dynamical_process(graph, S):
    inf_s = [set(S)]
    prev_inf_s = set(S)
     total_influenced = len(S) # Numero totale di nodi influenzati
    r = 0 # Contatore di iterazioni]

    while True:
        inf_r = set()
        for node in graph.nodes():
            if node not in prev_inf_s: # Se il nodo non è stato influenzato negli step precedenti
                neighbors = set(graph.neighbors(node))
                count_influenced_neighbors = len(neighbors.intersection(prev_inf_s))
                if count_influenced_neighbors >= len(neighbors) / 2:
                    inf_r.add(node)

        # Se non ci sono nuovi nodi influenzati, interrompi il processo
        if not inf_r:
            break

        # Aggiorna inf_s con l'insieme di nodi influenzati al passo corrente
        inf_s.append(inf_r)
        prev_inf_s = prev_inf_s.union(inf_r)
        total_influenced += len(inf_r) # Aggiorna il numero totale di nodi influenzati
        r += 1 # Incrementa il contatore di iterazioni

    return inf_s, total_influenced
```

## Sperimentazione

Per analizzare la bontà del nostro approccio al problema abbiamo svolto in totale 54 esecuzioni.

- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 1 con la funzione costi random
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 1 con la funzione Grado del Nodo/2
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 1 con la funzione costi basata sulla Centralità del Vicinato
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 2 con la funzione costi random
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 2 con la funzione Grado del Nodo/2
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 2 con la funzione costi basata sulla Centralità del Vicinato
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 3 con la funzione costi random
- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 3 con la funzione Grado del Nodo/2

- Sei esecuzioni con budget: 10, 25, 50, 75, 100 e 150 per la funzione 3 con la funzione costi basata sulla Centralità del Vicinato

Per ogni funzione abbiamo stampato un grafico dove l'ordinata è il numero di nodi influenzati dalla funzione scelta e l'ascissa è il budget utilizzato

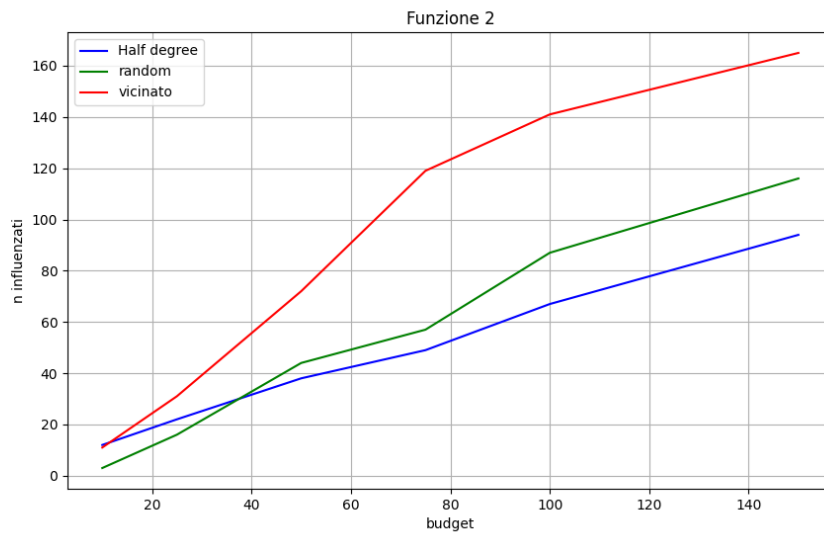


Figura 6 Funzione 1

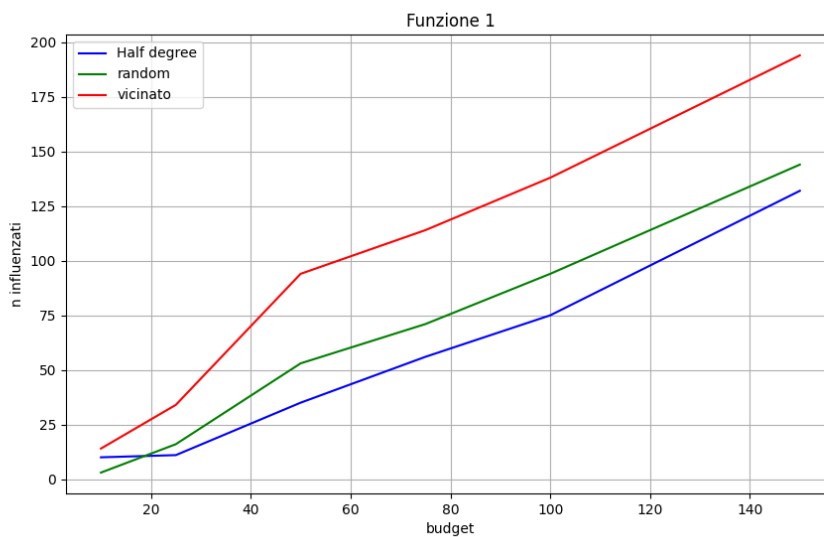


Figura 7 Funzione 2

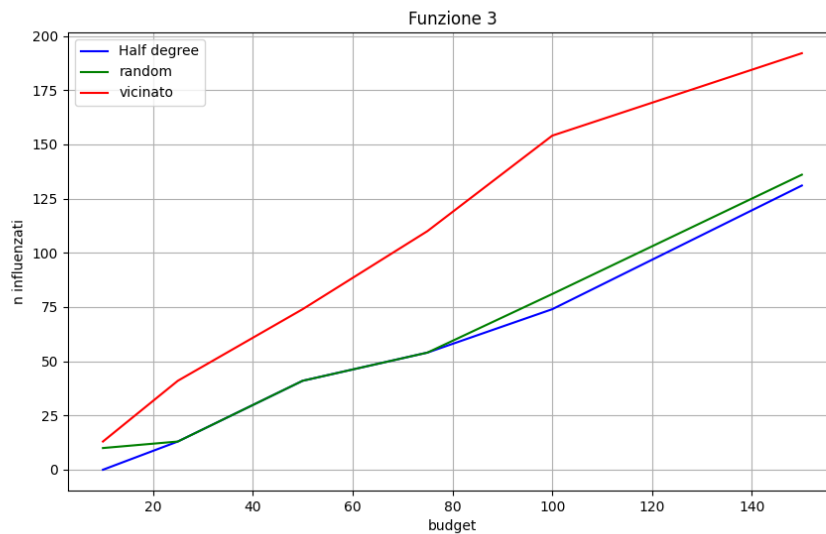


Figura 8 Funzione 3

## Conclusione

Le funzioni proposte riescono a influenzare un buon numero di nodi ma non riescono mai a generare una complete cascade, questo è dovuto al vincolo del budget per la scelta del Seed set e dal numero molto alto di cluster all'interno del grafo.

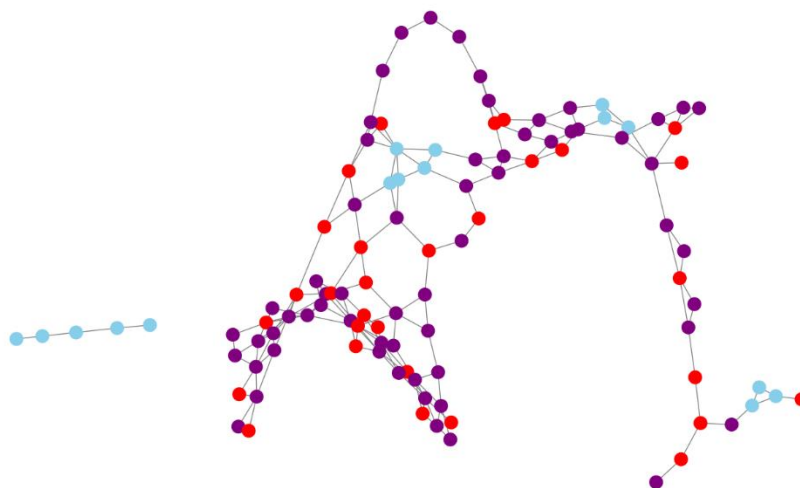
In tutte le esecuzioni utilizzando la nostra funzione costo otterremo non solo un Seed set più piccolo rispetto alle altre funzioni costo ma anche uno spread migliore dell'influenza.

Volendo visualizzare graficamente il risultato del processo di influenza avremo un grafo poco leggibile, per questo motivo abbiamo deciso di introdurre una seconda rete più piccola di nostra creazione, la rete "Provinceltaliane", dove ogni nodo è una provincia e ogni arco collega due nodi se le due provincie relative confinano.

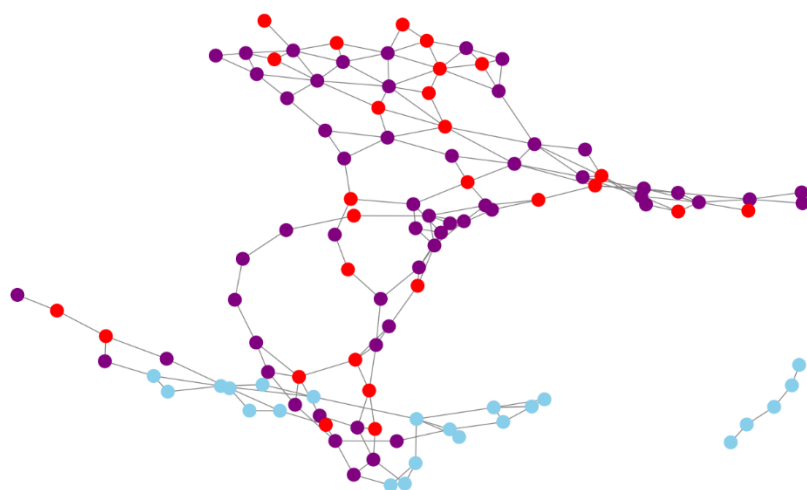
In totale ci sono 108 nodi all'interno della rete, utilizzando per tutte e tre le funzioni un budget 200 e la funzione costo centralità del vicinato (la più performante delle tre) e stampando il grafo evidenziando di rosso il seed set e di viola i nodi influenzati in step successivi otteniamo quindi:



*Figura 9 L'Italia usando la Funzione 1*



*Figura 10 l'Italia usando la Funzione 2*



*Figura 11l'Italia usando la Funzione 3*