Un approccio temporale per determinare la proprietà di notizie diffuse su Twitter

Candidato: Nicola Agresti
Matricola: VR407685
Relatore: Elisa Quintarelli
Correlatore: Sara Migliorini

Università degli Studi di Verona
Dipartimento di Informatica
Corso di laurea in Informatica

8 Ottobre 2020

Caratteristiche

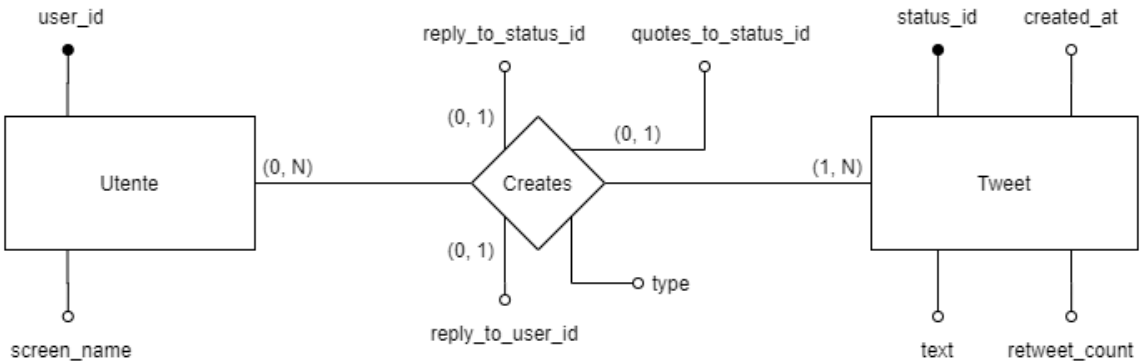
Alcune caratteristiche minime che deve possedere il dataset:

- Notizie da Twitter
- Autore
- Data
- Testo
- Presenza di retweet

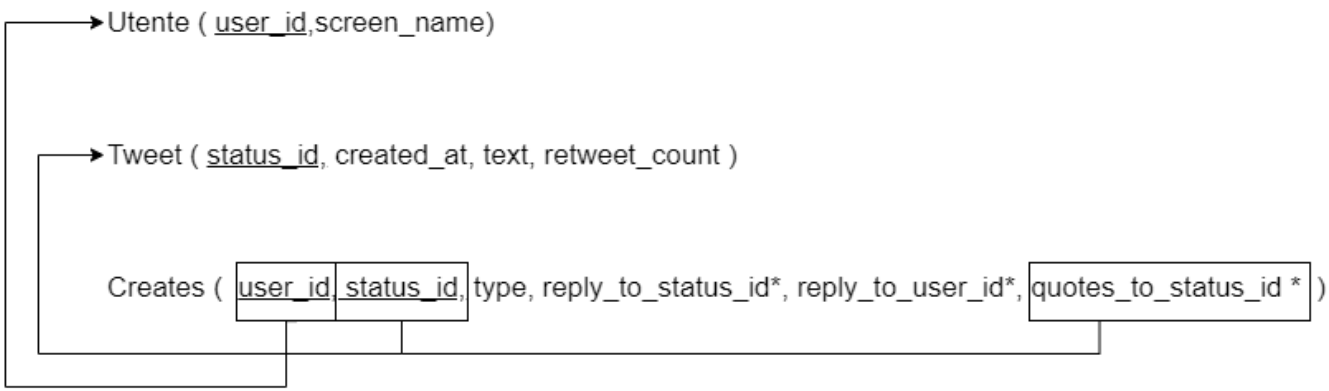
Queste sono necessarie per costruire uno schema relazionale corretto e successivamente le tabelle nella base di dati.

Progettazione Logica

Lo schema ER prodotto in base al problema è:



Lo schema relazionale quindi risulta:



Personalizzazione I

A partire dal dataset di partenza sono state eseguite delle modifiche per adattare le informazioni agli schemi prodotti dalla progettazione logica, in particolare sono stati creati 5 file di formato CSV.

Personalizzazione II

	A	B
1	user_id	screen_name
2	3171712086	RahulGandhi
3	93794912	seoulmania
4	705694814612938752	NCDCgov
5	102071743	GovWhitmer
6	16989178	JamesOKeefeIII
7	531041640	KKMPutrajaya
8	1030099481533014017	MazzoleniJulio
9	147994804	rishibagree
10	21059255	tedlieu

Figura: Esempio user.csv

Personalizzazione III

	A	B	C	D
1	status_id	created_at	text	retweet_count
2	1255701270901387264	30/04/2020 05:31	A conversation with Dr Raghuram Rajan, former RBI Governor, on dealing with the #Co	10146
3	1255852107673907202	30/04/2020 15:30	ì,¬íšŒì ê±°ë¦¬ëŠ” ë'ì§€ë§Œâ†"ë§ˆìŒì€ ê°€ê¹Œì› ìŠµë‹ˆë‹¤ðŸ'—#ìš°ë¦¬_ëª¨ë'ê°€_ì˜ì›…ìž	9521
4	1255993387603329026	01/05/2020 00:52	204 new cases of #COVID19 reported;80-Kano45-Lagos12-Gombe9-Bauchi9-Sokoto7-Bc	8852
5	1255689432738349059	30/04/2020 04:44	At 9 AM today, you can watch my conversation with Dr Raghuram Rajan, former RBI Go	8507
6	1255978307641905154	30/04/2020 23:52	Iâ€™ve said it before, and Iâ€™ll say it again â€" Michigan is an extraordinary place to l	6961
7	1255976688594419718	30/04/2020 23:45	UNBELIEVABLE! Our team just caught @Twitter red handed stealing retweets off our ne	5504
8	1255778934668398593	30/04/2020 10:40	Terkini 30 April #COVID19 ðŸ‡²ðŸ‡¾Kes baharu 5âƒ£7âƒ£ (25 kes import)Kematian	4629
9	1255682896020541440	30/04/2020 04:18	Informe #COVID19: hoy procesamos 563 muestras, 10 dieron positivo: 5 del exterior, 1	4400
10	1255857575783919617	30/04/2020 15:52	Didi Stays in a Gated Community at 41st floor.Now tell me which beggar/ Unknown per	4239

Figura: Esempio tweet.csv

Personalizzazione IV

	user_id	status_id	type	reply_to_status_id	reply_to_user_id
28	921525775169458176	1255968042787524609	TW		
29	22703339	1255922986726707200	TW		
30	109263459	1255909313316601858	RE	1255905597976969216	109263459
31	16910746	1255860128231895044	TW		
32	11347122	1255949158751571968	TW		
33	705694814612938752	1255997250544812035	RE	1255993387603329026	705694814612938752
34	77372998	1255852155967332352	TW		
35	12272322	1255930519298224134	TW		
36	42407972	1255981948872413184	TW		
37	1154180084078501889	1255947666866843648	QT		
38	928677206	1255894000827011072	TW		

Figura: Esempio creates.csv

Personalizzazione V

	user_id	status_id	type	reply_to_status_id	reply_to_user_id	retweet_count
28	921525775169458176	1255968042787524609	TW			1947
29	22703339	1255922986726707200	TW			1691
30	109263459	1255909313316601858	RE	1255905597976969216	109263459	1617
31	16910746	1255860128231895044	TW			1616
32	11347122	1255949158751571968	TW			1598
33	705694814612938752	1255997250544812035	RE	1255993387603329026	705694814612938752	1591
34	77372998	1255852155967332352	TW			1563
35	12272322	1255930519298224134	TW			1443
36	42407972	1255981948872413184	TW			1429
37	1154180084078501889	1255947666866843648	QT			1428
38	928677206	1255894000827011072	TW			1428

Figura: Esempio creates_con_rt.csv con numero dei retweet

Il dataset non conteneva le istanze dei retweet (type = RT), di conseguenza sono state generate tramite un programma creato appositamente, il quale farà uso di questo file.csv.

Personalizzazione VI

Codice 1: Porzione di codice usato per generare i retweet

```
for row in reader:
        if row['retweet_count'] > 0:
            for tupla in range(row['retweet_count']):
                stringa = {"user_id": user[posUtente],
                           "status_id": row['status_id'],
                           "type": "RT",
                           "reply_to_status_id": None,
                           "reply_to_user_id": None}
                retweet.append(stringa)
                if posUtente != len(user) - 1:
                    posUtente = posUtente + 1
                else:
                    posUtente = 0
```

Questo programma crea N istanze di retweet (inventate) per ogni tweet che ha almeno un retweet, salvando il tutto in un file.json.

Creazione

Il database è stato creato localmente sul PC con PostgreSQL.

Codice 2: Query usata per generare le tabelle

```
CREATE TABLE utente(
    user_id VARCHAR PRIMARY KEY CHECK (user_id <> ''),
    screen_name VARCHAR NOT NULL CHECK (screen_name <> '')
);
CREATE TABLE tweet(
    status_id VARCHAR PRIMARY KEY CHECK (status_id <> ''),
    text VARCHAR NOT NULL CHECK (text <> ''),
    created_at TIMESTAMP NOT NULL CHECK (created_at <= CURRENT_TIMESTAMP),
    retweet_count INTEGER CHECK (retweet_count >= 0)
);
CREATE DOMAIN TYPETWEET VARCHAR(4) CHECK(
    VALUE IN ('TW', 'RE', 'RT', 'QT', 'REQT')
);
CREATE TABLE creates(
    user_id VARCHAR REFERENCES utente,
    status_id VARCHAR REFERENCES tweet,
    type TYPETWEET NOT NULL,
    reply_to_status_id VARCHAR DEFAULT NULL,
    reply_to_user_id VARCHAR DEFAULT NULL,
    quotes_to_status_id VARCHAR DEFAULT NULL REFERENCES tweet,
    PRIMARY KEY (user_id, status_id)
);
```

Upload I

Per eseguire l'upload ho usato Python e le conoscenze apprese durante il corso di laboratorio di Basi di Dati.

```
1  import json
2  import psycopg2
3  from myAppConfig import myHost, myDatabase, myUser, myPsw
4
5  connessione = psycopg2.connect (host = myHost, database = myDatabase, user =
       myUser, password = myPsw)
6
7  with connessione:
8      with connessione.cursor() as cursore:
9
10         with open("createsRT.json", encoding='utf8') as f:
11             data = json.load(f)
12
13         keys = []
14         for row in data:
15             for key in row.keys():
16                 if key not in keys:
17                     keys.append(key)
18
19         for row in data:
```

Upload II

```
20              query = "INSERT INTO creates (user_id, status_id, type,
                    reply_to_status_id, reply_to_user_id)
                    VALUES({0});".format(",".join(map(lambda key:
                    "'{0}'".format(row[key]) if key in row else "NULL", keys)))
21              cursore.execute(query)
22
23          print("Caricamento completato, tuple inserite!")
24
25  connessione.close()
26  print("Connessione chiusa con successo!")
```

Modificando il testo della query si può adattare questo programma per popolare qualsiasi tabella.

Struttura I

I nodi del grafo:

- DB: la radice del grafo
- Tweet: rappresenta un' istanza di un tweet
- Retweet: rappresenta un' istanza di retweet di un nodo tweet

Gli archi del grafo:

- DB $\rightarrow$ Tweet
- DB $\rightarrow$ Retweet
- Tweet $\rightarrow$ Retweet
- Tweet $\rightarrow$ Tweet successivo (stesso creatore)
- Retweet $\rightarrow$ Tweet successivo (se presente)
- Tweet precedente $\rightarrow$ Retweet (se presente)

Struttura II

Ogni arco è descritto da un'etichetta
$< Start, End, Author, Weight >$, dove Weight è:

- 1 se l'arco è entrante in un Tweet
- 0.1 se l'arco è entrante in un Retweet
- 0.5 se l'arco è entrante in un Tweet di tipo quotes

Start/End rappresentano un intervallo di minuti che descrive l'ipotetica creazione del Tweet o Retweet (a partire da un determinato istante), Author è il proprietario del Tweet o Retweet.

Struttura III

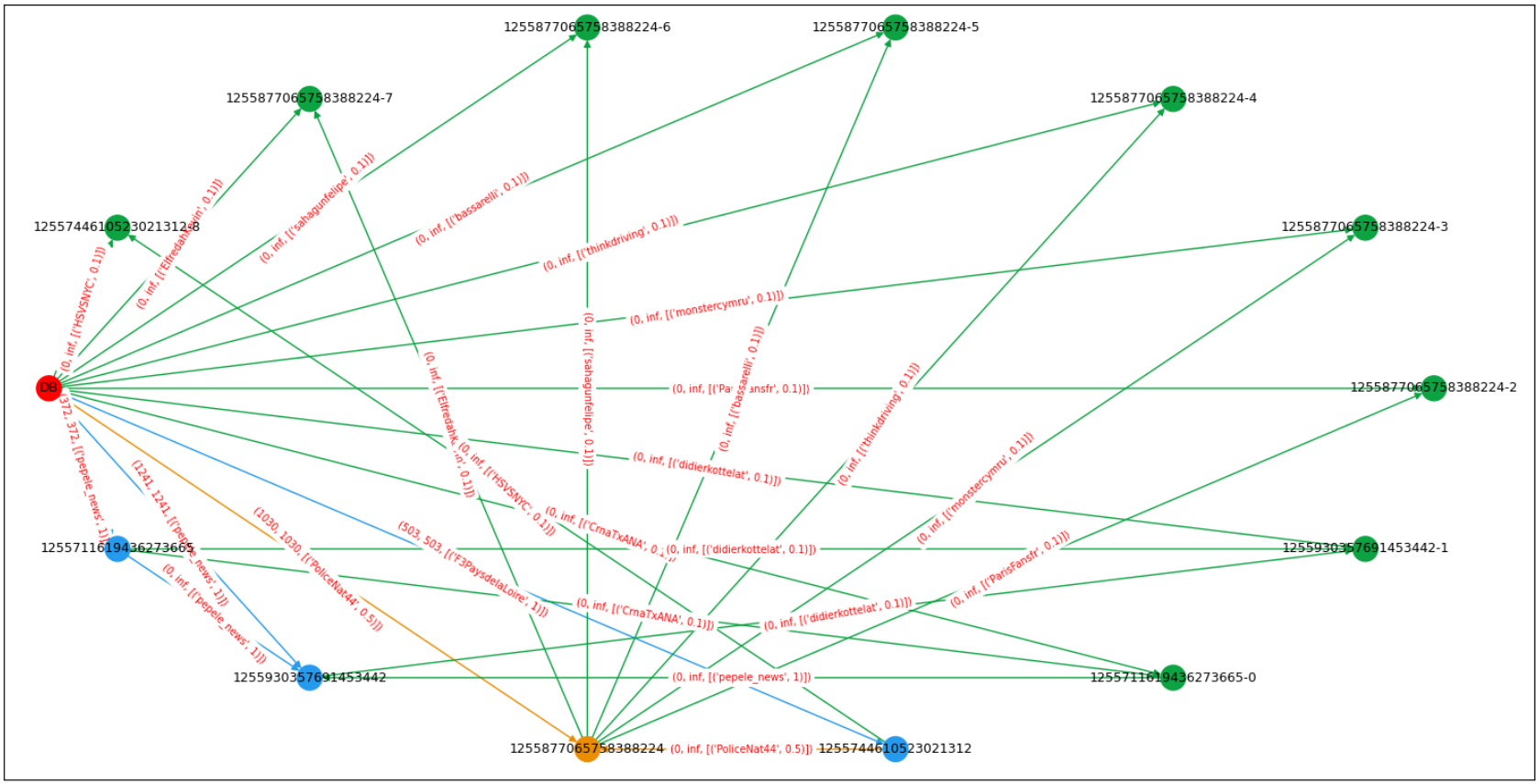


Figura: Grafo di esempio

Creazione I

Per creare il grafo precedente ho usato Python e la libreria NetworkX, la quale permette molte personalizzazioni e una gestione del grafo, tramite i metodi della libreria, molto efficace.

Per ottenere i nodi necessari sono state implementate nel codice alcune query che ora vedremo.

Creazione II

Codice 3: Query per ottenere i Tweet

```
1  DROP VIEW IF EXISTS user_id_quotes;
2  CREATE VIEW user_id_quotes AS (
3      SELECT U.screen_name, U.user_id, T.status_id
4      FROM utente U
5          JOIN creates C ON U.user_id = C.user_id
6          JOIN tweet T ON C.status_id = T.status_id
7      WHERE C.type = 'TW' AND
8          T.status_id IN (SELECT Ci.quotes_to_status_id
9                          FROM creates Ci
10                         WHERE Ci.quotes_to_status_id IS NOT NULL)
11                         );
12 SELECT T.status_id, T.created_at, U.screen_name, U.user_id, C.type,
       C.quotes_to_status_id, V.screen_name AS authorTW
13 FROM tweet T
14     JOIN creates C ON T.status_id = C.status_id
15     JOIN utente U ON C.user_id = U.user_id
16     LEFT JOIN user_id_quotes V ON C.quotes_to_status_id = V.status_id
17 WHERE C.type <> 'RT' AND
18       T.status_id IN ('1255930357691453442')
19 ORDER BY U.user_id, T.created_at;
```

Creazione III

Codice 4: Query per ottenere i Retweet

```
1  DROP VIEW IF EXISTS User_Id_Originale;
2  CREATE VIEW User_Id_Originale AS (
3      SELECT U.user_id, T.status_id, U.screen_name
4      FROM tweet T
5          JOIN creates C ON T.status_id = C.status_id
6          JOIN utente U ON C.user_id = U.user_id
7      WHERE C.type <> 'RT' AND
8          T.status_id IN ('1255930357691453442')
9      ORDER BY U.user_id
10 );
11 SELECT U.user_id, T.status_id, T.created_at, U.screen_name, V.user_id,
       V.screen_name
12 FROM utente U
13     JOIN creates C ON U.user_id = C.user_id
14     JOIN tweet T ON C.status_id = T.status_id
15     JOIN User_Id_Originale V ON T.status_id = V.status_id
16 WHERE C.type = 'RT' AND
17       T.status_id IN ('1255930357691453442')
18 ORDER BY V.user_id, T.created_at;
```

Creazione IV

Dopo aver ottenuto i nodi si passa alla creazione degli archi con etichette. Questo è possibile tramite il metodo *Graph.add_edge*(*nodo*1, *nodo*2, *attributi*), per esempio:

Codice 5: Porzione di codice dove aggiungo un arco con etichetta

```
1 attr_dict = {}
2 attr_dict["Start"] = 0
3 attr_dict["End"] = math.inf
4 attr_dict["Authors"] = [(creator_screen_name[posTW+1], 1)]
5 stringa = tuple(x for x in attr_dict.values())
6 time_list = []
7 time_list.append(stringa)
8 graph.add_edge(n, tweet[posTW+1], attr_dict=attr_dict, time=stringa,
      time_list=time_list)
```

Ogni nodo quindi sarà connesso con un altro secondo la struttura del grafo spiegata in precedenza.

Definizioni I

Inversione

Data una tupla $T_k =< Start, End >$ il suo inverso è $T_k^{-1} =< -End, -Start >$

Esempio

$R_{ij} = (200, 500) \rightarrow \mathrm{R}_{ij}^{-1} = (-500, -200)$

Definizioni II

Composizione ∘

Date due tuple $T_{k1} =< Start_1, End_1 >$ e $T_{k2} =< Start_2, End_2 >$ la composizione è
$T_{k12} =< Start_1 + Start_2, End_1 + End_2 >$, in pratica la somma di due tuple.

Esempio

$R_{ij1} = (200, 500)$ e $R_{ij2} = (300, 100) \rightarrow \mathrm{R}_{ij12} = (500, 600)$

Definizioni III

Congiunzione $\otimes$

Date due tuple $T_{k1} =< Start_1, End_1 >$ e $T_{k2} =< Start_2, End_2 >$ la congiunzione è
$T_{k12} =< MAX(Start_1, Start_2), MIN(End_1, End_2) >$, in pratica crea una tupla con il valore Start massimo e il valore End minimo tra le due tuple.

Esempio

$R_{ij1} = (\infty, 100)$ e $R_{ij2} = (350, 50) \rightarrow R_{ij12} = (\infty, 50)$

Definizioni IV

Path-Consistency

Date tre tuple $R_{ij} =< Start, End >$, $R_{ik} =< Start, End >$ e $R_{kj} =< Start, End >$ l'algoritmo esegue quanto segue

$$R'_{ij} = R_{ij} \otimes (R_{ik} \circ R_{kj})$$

Esempio

$$R_{ij} = (0, \infty),\ R_{ik} = (372, 372),\ R_{kj} = (0, \infty)$$
$$R'_{ij} = (0, \infty) \otimes ((372, 372) \circ (0, \infty))$$
$$R'_{ij} = (0, \infty) \otimes (372, \infty)$$
$$R'_{ij} = (372, \infty)$$

Può accadere che in alcuni casi di studio serva applicare l'inversione a una tupla per ottenere un risultato coerente con il grafo.

Definizioni V

Intersezione Intervalli $\cap$

Dati due intervalli $T_1 =< Start_1, End_1 >$ e $T_2 =< Start_2, End_2 >$ l'intersezione $T_{12} = T_1 \cap T_2$ è data dal valore minimo di Start $min(Start_1, Start_2)$ e quello massimo di End $max(End_1, End_2)$, nel caso End sia maggiore o uguale di Start il risultato è la sottrazione $End - Start$ altrimenti 0.

Esempio

$T_1 =< 372, 372 >,\ T_2 =< 550, 550 >$
$T_{12} = T_1 \cap T_2$
$min(Start_1, Start_2) = 372$
$max(End_1, End_2) = 550$
$T_{12} = 178$

Definizioni VI

Unione Intervalli $\cup$

Dati due intervalli $T_1 =< Start_1, End_1 >$ e $T_2 =< Start_2, End_2 >$ l'unione $T_{12} = T_1 \cup T_2$ è data dal valore massimo di Start $max(Start_1, Start_2)$ e quello minimo di End $min(End_1, End_2)$, nel caso End sia maggiore o uguale di Start il risultato è la sottrazione $End - Start$ altrimenti 0.

Esempio

$T_1 =< 372, 372 >,\ T_2 =< 550, 550 >$
$T_{12} = T_1 \cup T_2$
$max(Start_1, Start_2) = 550$
$min(End_1, End_2) = 372$
$T_{12} = 0$

Definizioni VII

Similarità

Dati due intervalli $T_1 =< Start_1, End_1 >$ e $T_2 =< Start_2, End_2 >$ la similarità è data da $sim(T_1, T_2) = \frac{T_1 \cap T_2}{T_1 \cup T_2}$, nel caso l'unione risulti infinita il risultato è 0.1 (poca similarità), nel caso sia diversa da 0 il risultato è dato dal rapporto tra l'intersezione e l'unione, altrimenti risulta 0.

Esempio

$T_1 =< 372, 372 >,\ T_2 =< 550, 550 >$
$T_{1 \cap 2} = 178$
$T_{1 \cup 2} = 0$
$sim(T_1, T_2) = 0$

Applicazione

Per applicare in modo efficace l'algoritmo al grafo sono state individuate tutte le combinazioni tra nodi e archi che formano un triangolo. In particolare è stata usata questa funzione:

Codice 6: Funzione che ritorna i triangoli formati da 3 nodi

```
1  def get_triangles(G, i, j):
2      result = []
3      for nodo in G.nodes():
4          if (G.has_edge(i, nodo) and G.has_edge(nodo, j)) or \
5             (G.has_edge(i, nodo) and G.has_edge(j, nodo)) or \
6             (G.has_edge(nodo, i) and G.has_edge(nodo, j)) or \
7             (G.has_edge(nodo, i) and G.has_edge(j, nodo)):
8
9              triangle = (i, nodo, j)
10             result.append(triangle)
11
12     return result
```

Successivamente ad ogni triangolo è stato applicato l'algoritmo Path-Consistency e il calcolo della similarità tra gli intervalli.

Esempio I

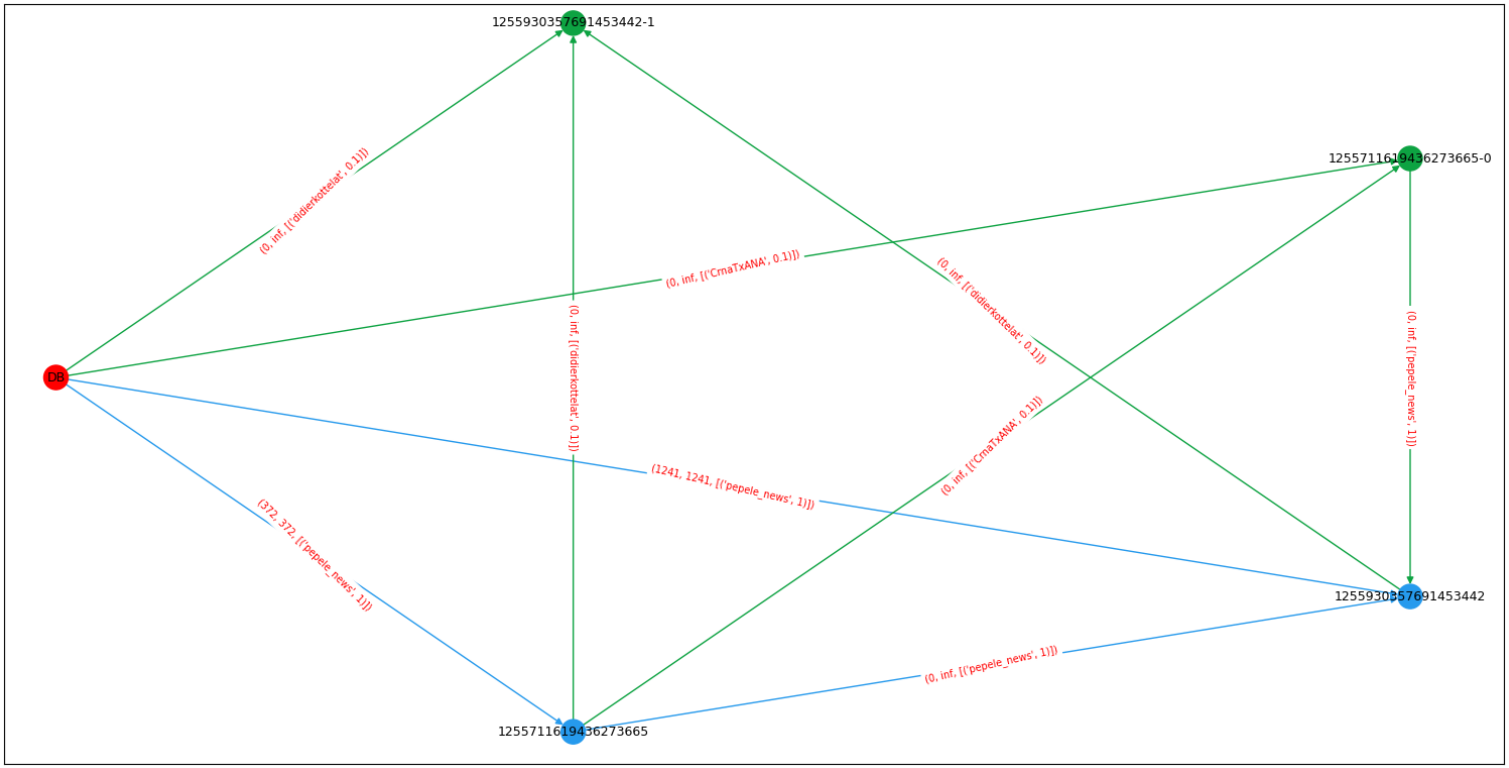


Figura: Grafo senza algoritmo path-consistency applicato

Esempio II

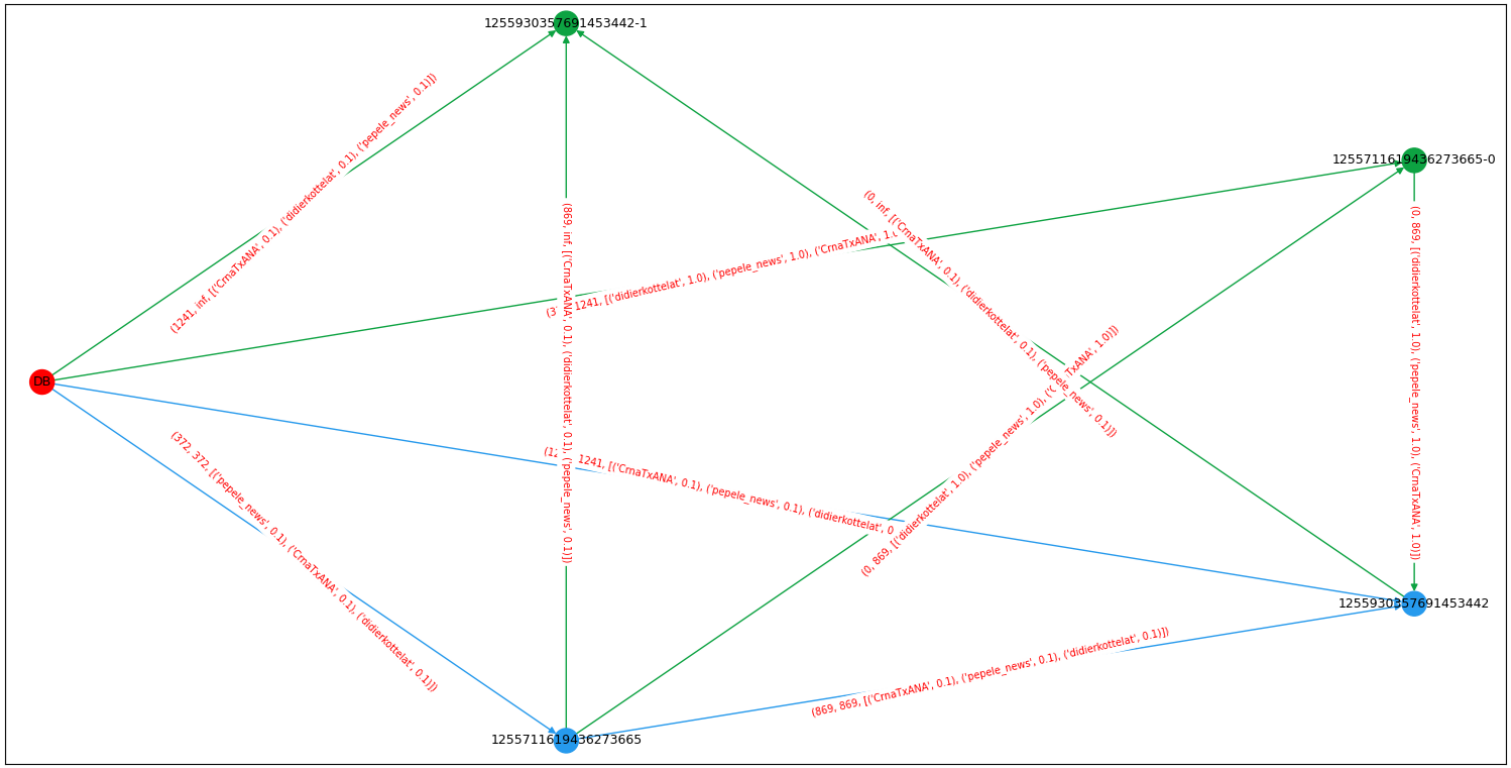


Figura: Grafo con algoritmo path-consistency applicato


Grazie dell'attenzione