ALGORITMO PER L'ANALISI E LA CREAZIONE DEL GRAFO DEI RIFERIMENTI GIURIDICI SUL SITO ISTITUZIONALE "NORMATTIVA"

Fabio Pisano*

SOMMARIO: 1.- Introduzione; 2.- Revisione della letteratura; 3.- Limiti di questo approccio e parametri; 4.- Risultato; 5.- Descrizione dell'algoritmo; 6.- Possibili Sviluppi; 7.- Appendice.

1.- **Introduzione.**

Esistono norme giuridiche il cui significato non si esaurisce all'interno dell'enunciato stesso, ma all'interno di altre norme. I riferimenti possono essere impliciti (ad esempio facendo riferimento ad un istituto giuridico o ad una fattispecie giuridica) o espliciti (facendo esplicito riferimento alla norma giuridica a cui si rimanda). Il secondo tipo di riferimenti può essere svolto in maniera meccanica, in quanto non richiede nessun tipo di interpretazione, ma soltanto far combaciare degli identificativi. Se il significato di una espressione dipende da un'altra, avendo a disposizione entrambe le espressioni (quella di partenza e quella di riferimento) e collegandole opportunamente attraverso espliciti riferimenti, sarà possibile avere una nuova espressione che non fa esplicito riferimento ad altro. In tal modo sarà possibile ottenere un testo giuridico esaustivo, per lo meno dal punto di vista dei riferimenti espliciti.

Il sito istituzionale normattiva.it è strutturato in maniera tale da poter svolgere un tipo di analisi del genere. Il testo di una norma è redatto attraverso il protocollo HTML, un tipo di scrittura di testo dinamica (corrispondente al paradigma WYSIWYM)[1] ovverosia un documento il cui testo inserito non corrisponde al testo finale. All'interno di questo sarà pertanto possibile identificare il nodo il quale include il corpo del testo della norma giuridica. A questo punto, all'interno di questo testo residuale possiamo identificare tutti i collegamenti ipertestuali che rimandano ad altre disposizioni, come ad esempio[2]:

```
<a href="/uri-res/N2Ls?urn:nir:stato:costituzione:1947-12-27~art87-com5"
target="_blank">articolo 87, quinto comma, della Costituzione</a>
```

Si può ripetere la medesima operazione per le nuove norme che sono state identificate fino a che non sarà possibile andare oltre, ottenendo non soltanto tutte le norme, ma anche l'ordine di apparizione delle medesime che può agevolmente essere inserito in un grafo ad albero. É possibile implementare queste operazioni in un codice informatico che svolga automaticamente questo tipo di lavoro, tale per cui, una volta inserito l'URL associato alla norma giuridica iniziale nella funzione, il computer sia in grado di svolgere tutte le operazioni di lettura ed ordinamento che sono state descritte. É stato sviluppato un algoritmo scritto attraverso il linguaggio Python (scritto nella versione 3.8.18, testato anche in 3.11) la cui versione integrale è resa disponibile nell'appendice di questo articolo. É possibile inoltre offrire una valutazione quantitativa dell'efficienza di questa metodologia confrontando, per

* Dottore di Ricerca, presso Pegaso International, H.E.I., Valletta, Malta.

[1] What You See Is What You Mean.

[2] Un attento osservatore potrà notare come in questo caso, il collegamento ipertestuale è troncato, in quanto non viene esplicitato nel corpo del testo la radice: https://www.normattiva.it.
È stato, pertanto, necessario gestire le circostanze in cui il collegamento esplicitato contenesse o meno il riferimento alla pagina principale.

ogni iterazione, il numero di riferimenti che sono stati registrati nell'output e il numero di riferimenti che sono stati registrati in totale (includendo di conseguenza quelli esclusi). Rapportando queste due grandezze si può agevolmente osservare che esse rappresentano due funzioni monotone crescenti, le cui derivate corrispondono a 0 quando l'algoritmo cessa di funzionare.
Utilizzando come punto di partenza una norma differente, il risultato sarà diverso. Infine, si è tentato di implementare un meccanismo che possa permettere di evitare i riferimenti circolari, fenomeno che, tuttavia, viene sostanzialmente imputato alla struttura del sito internet e non ad una eventuale incoerenza del sistema giuridico rappresentato. Si è ritenuto opportuno redigere questo articolo in lingua italiana poiché, analizzando un sito che tratta norme italiane, i portatori di interessi al di fuori dell'ambito puramente accademico sono soggetti che utilizzano questa lingua. La sperimentazione è stata testata sulla seguente norma[3]: D.P.R. 16/04/2013, n. 70.

2.- **Revisione della letteratura.**

G. Perigunelli, M. Ragona, *50 anni di studi, ricerche ed esperienze, L'Informatica Giuridica in Italia* (2014) Appendice 2 rappresenta il punto di partenza per comprendere gli sforzi intrapresi a partire dagli anni 70 in Italia per l'utilizzo dei computer nel diritto. In particolar modo, è importante sottolineare come gli sforzi non siano stati puramente accademici, ma anche Senato e Camera dei Deputati abbiano intrapreso iniziative per costituire basi dati pubbliche[4]. L'allegato 2 della medesima pubblicazione fornisce altresì una bibliografia dei principali lavori di informatica giuridica in Italia.
Il progetto Normattiva.it[5] si inserisce in questa tradizione e nasce all'inizio del XXI secolo.
Sono state effettuate alcune sperimentazioni su Normattiva.it[6].

3.- **Limiti di questo approccio e parametri.**

Il limite di questa metodologia si basa sull'assunto fondamentale che i riferimenti giuridici validi siano tutti ed i soli collegamenti ipertestuali identificati. Ciò non è vero, si sono riscontrati alcuni limiti:

- Falsi Positivi, inclusione di riferimenti non corretta;
- Falsi Negativi, omissione di norme effettivamente presenti.

Tutte le norme che non sono contenute nel sito, come ad esempio le norme di diritto comunitario, oppure di diritto internazionale e norme di diritto nazionale non indicizzate all'interno del sito (ad esempio delle pubblicazioni in Gazzetta Ufficiale) non vengono approfondite da questo algoritmo: Questo algoritmo, infatti, è in grado di identificarle nel corpo del testo, ma l'assenza dei collegamenti ipertestuali non permette di identificare gli eventuali riferimenti espliciti che queste norme possiedono. Nella sezione FAQ, il sito Normattiva.it esplicita alcune regole alla voce "Come si fa a linkare gli atti di "Normattiva"?"[7], in particolar modo:

[3] La scelta di questa norma in particolare mi è stata suggerita da una collega del mio corso di Dottorato. L'argomento di questa norma riguarda la scuola.
[4] Idd., *50 anni di studi, ricerche ed esperienze, L'Informatica Giuridica in Italia*, Roma, 2014, Parte Seconda, Capitolo III – IV.
[5] https://www.normattiva.it/staticPage/progettoCoordinamento.
[6] Affrontano il problema di internalizzare le differenti versioni di una norma giuridica all'interno del formato HTML i seguenti studi: A. Ciaghi, A., Dalla Valle, A. Villafiorita, A, *Adapting Software Metrics to Analyze the Evolution of Laws* in *Frontiers in Artificial Intelligence and Applications* (2011) 53-62, N. Lettieri, A. Altamura, D. Malandrino, *The legal macroscope: Experimenting with visual legal analytics. Information Visualization* 16(4) (2017) 332-345, F. Stellato, *LegalHTML: Semantic mark-up of legal acts using web technologies* 51 (2023).
[7] https://www.normattiva.it/staticPage/faq#9.

- @originale ed !vig=, queste stringhe di testo, aggiunte opportunamente a ciascun URL permettono rispettivamente di ottenere la versione originale oppure la versione in vigore ad oggi (o ad una determinata data) di un testo;
- ~, questa stringa permette di ottenere esclusivamente un articolo (o un comma) di un testo giuridico.

Pertanto, come viene specificato nel sito internet, un collegamento del genere https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:decreto.legge:2008-11-10;180!vig=2009-11-10 contiene un testo all'interno del quale i riferimenti sono successivi a Novembre 2008 e precedenti a Novembre 2009 e potrebbe essere auto-referenziale. Ad esempio, se si utilizza l'URL della Costituzione https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:costituzione~art26 per identificare Cost. it., art. 26, la versione contemporanea fa riferimento ad un'altra norma, L. Cost 21/06/1967, n. 1 che, a sua volta, si riferisce ad una versione precedente della norma iniziale Cost. it., art. 26, 27/12/1947 individuata dall'URL
https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:costituzione:1947-12-27~art26
che, comunque, include nel corpo del testo il riferimento alla norma successiva.

I riferimenti possono essere di una norma giuridica o di una parte di essa, distinzione che si può identificare se nel collegamento ipertestuale appare il simbolo tilde ~ (che separa il nome della norma giuridica dall'eventuale articolo e comma). Un riferimento ad una norma giuridica in generale e non ad uno specifico articolo determina un errore di falso positivo. Per gestire la contemporanea presenza di collegamenti ipertestuali rispettivamente di una norma o di una sua componente, è necessario tenere in considerazione che ciò può inficiare la dinamica dell'efficienza dell'algoritmo medesimo. Se, per ogni iterazione, risulta possibile espungere alcuni riferimenti che sono stati tenuti distinti poichè rappresentano degli articoli (o dei commi) della medesima norma giuridica generale i cui riferimenti giuridici siano identici, sarà possibile avere una dinamica decrescente nel numero di riferimenti registrati nell'output. Per affrontare questo problema, si è ritenuto opportuno creare due nuove colonne nelle tabelle di output:

- URL_originale: che corrisponde alla versione integrale di un URL;
- URL: che corrisponde a quanto effettivamente trovato.

Pertanto, possono esistere differenti URL associati allo stesso URL_originale, in quanto parti dello stesso atto giuridico. In tal caso, tutte le volte che un nuovo URL viene letto, per essere accettato, ciascun elemento della lista di output dovrà rispettare le seguenti condizioni:

a) Sia l'elemento, sia URL_originale associabile non devono comparire nella lista degli URL già individuati;
b) Gli URL_originale associati ai nuovi elementi che ne siano diversi non appaiano nella lista degli elementi di input.

In tal modo:

a) si eliminano non solo i duplicati, ma anche i duplicati parziali di una norma generale già individuati;
b) Nel caso in cui l'url di input abbia al suo interno i riferimenti ad una sottonorma e alla norma generale, solo la norma generale viene registrata.

La numerazione delle nuove norme, avviene attraverso un identificativo di questa natura

- $X.n_1.n_2. \ldots .n_n(T)$
- $X.n_1.n_2. \ldots .n_n(s_{a,i}\_s_{c,i})$

Tale per cui:

- La numerosità dei punti, rappresenta il livello di profondità dell'albero (a partire dal riferimento giuridico iniziale identificato con X) nel quale si colloca il nuovo riferimento;
- Ciascun sottoriferimento corrisponde al formato n_n che può semplicemente contenere un numero intero (se il riferimento associato corrisponde ad una norma), oppure ad $n_n(s_{a,i}\_s_{c,i})$ (se il riferimento associato corrisponde ad una parte della norma stessa, associata al numero di articolo $s_{a,i}$ ed al numero di comma $s_{c,i}$

Pertanto, ciascun identificativo in input, genererà un numero di nuovi identificativi pari al numero di nuovi URL accettati, la nuova numerazione:

- $X.n_1.n_2. \quad .n_n.n_{n+1}(s_{a,i}\_s_{c,i})$

Una problematica legata a questo tipo di numerazione risiede quando nel ciclo vengono individuati, non consecutivamente:

a) due sotto-norme della medesima norma;
b) prima una sotto-norma, e successivamente la norma generale.

In entrambi i casi i riferimenti vengono associati a due identificativi differenti, pertanto all'interno del grafo ad albero verrebbe a perdersi la relazione che esiste tra di loro. La relazione, tuttavia, si perde esclusivamente nel tipo di grafo che si sta studiando, in quanto l'identificativo così costituito non tiene conto degli URL originale (la variabile condivisa in entrambi i casi sopraesposti). L'unica implementazione che si propone è l'indicatore:

- X. $.n_{n+1}$(T), corrisponde ad aver individuato una norma integrale;
- X. $.n_{n+1}$(C), corrisponde ad aver individuato una norma integrale dopo aver già individuato una sua sotto-norma, in tal caso, pertanto, i riferimenti registrati saranno soltanto quelli complementari;
- X. $.n_{n+1}$(E), corrisponde ad isolare i riferimenti a siti esterni (qualunque essi siano)[8].

Sarà pertanto possibile utilizzare un colore differente per il nodo che viene illustrato (ad esempio grigio) rispetto a quelli trovati. In tal modo, dovrebbe essere garantita l'assenza di ciclicità nei riferimenti giuridici ed i collegamenti ipertestuali di "Normattiva.it" possono costituire una versione esaustiva di un testo[9]. Inoltre, l'algoritmo di ricerca così sviluppato potrà godere delle seguenti proprietà matematiche:

1) L'andamento dell'output parziale sarà una funzione monotona crescente, pertanto, una volta che un nuovo riferimento normativo è stato aggiunto, non può essere eliminato;
2) L'andamento dei risultati parziali ottenuti in ciascuna singola iterazione non sarà mai minore dei risultati che contribuiscono all'output definitivo.

In tal caso, il livello di inefficienza di questo metodo di ricerca potrà essere studiato dall'area compresa tra queste due funzioni (che rappresenta la quantità di riferimenti ad altre fonti di ciascuna norma analizzata non inserita nel grafo), rappresentabile con questo semplice integrale:

$$\int_0^n [INDIVIDUATI(x) - TRATTENUTI(x)]dx$$

[8] Sarebbe in questo caso possibile procedere ulteriormente con la classificazione, ad esempio (G) per i riferimenti dalla Gazzetta Ufficiale, oppure (EU) per i riferimenti del diritto Comunitario.

[9] Anche se tecnicamente possibile (ad esempio con espressioni del tipo "ai sensi di quanto sarà definito nel nuovo articolo X del codice Y") è invece da ritenersi poco probabile una reale *petitio principii*. Una norma può fare esplicito riferimento soltanto ad una norma pre-esistente, non ad una norma successiva. Una "riserva di legge", infatti, prescrive che esisteranno in futuro delle norme (le quali esse faranno riferimento a questa prima norma) ma non si riferisce a queste in maniera esplicita, in quanto non esistono ancora.

4.- **Risultato.**

Il risultato di questa operazione corrisponderà ad un grafo ad albero: partendo dal vertice iniziale (corrispondente alla norma di partenza) sarà quindi possibile rappresentare tanti archi quante sono le norme a cui fa riferimento, e ripetere questa medesima operazione per tutti i nuovi livelli identificati, fino ad arrivare a conclusione.

Questo tipo di grafo è poco funzionale per norme generali, mentre molto più utile per norme particolari, non è un caso che i grafi associati a ciascun articolo della Costituzione Italiana siano poco complessi, proprio perché è più facile che ci siano norme che fanno riferimento a queste ultime, mentre è molto più raro che queste norme facciano riferimento ad altre.

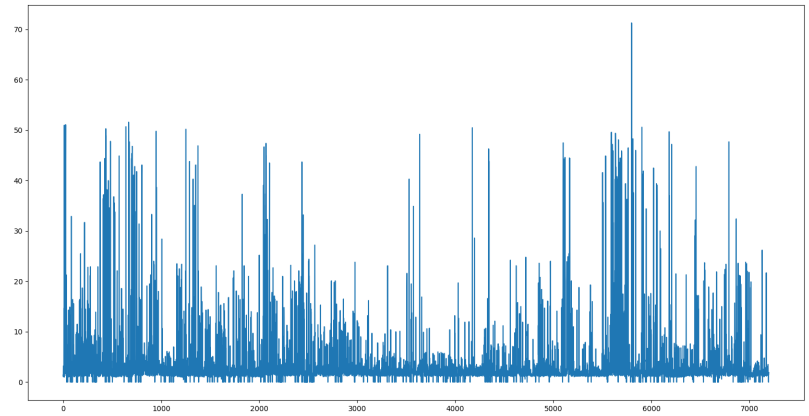
Infatti, possiamo controllare attraverso questo semplice ciclo di iterazione:

```
for i in range(1,139):
    url = 'https://www.normattiva.it/uri-res/N2Ls?urn:nir:stato:costituzione:1947-12-27'
    url_c = url + '~art' + str(i)
    OUTPUT_C, ATTESA_C = CICLO_GENERALE(url_c)
    print(f"ARTICOLO no. {i}, HA {OUTPUT_C.shape[0] - 1} RIFERIMENTI GIURIDICI")
```

Solo gli articoli 10, 26, 56, 57, 60, 81, 97, 117, 119 contengono qualche riferimento ulteriore (delle riforme di legge costituzionale).

Un ulteriore limite di questa rappresentazione corrisponde al fatto che una medesima norma possa apparire più volte. Da un punto di vista giuridico, una circostanza del genere non deve sorprendere: Una norma generale di una qualsiasi disciplina può avere tante norme particolari a cui fanno riferimento, anche a più livelli. La rappresentazione del grafo potrebbe risultare inutilmente profonda, non solo rallentando le prestazioni di esecuzione del computer, ma anche mostrando un grafo con un grado di complessità maggiore rispetto a quello che è nella realtà. Si è preferito favorire la velocità di esecuzione rispetto alla significatività del collegamento: nel caso di molteplici occorrenze di un medesimo riferimento giuridico, la norma è registrata nel grafo solo la prima volta in cui viene incontrata, mentre le successive occorrenze vengono escluse. Infine, l'algoritmo così sviluppato non ha grandi prestazioni: prendendo come riferimento 1, 2 o 3 secondi per ciascuna iterazione: avendo svolto, approssimativamente, 50 mila cicli, un codice del genere impiegherebbe, rispettivamente 13, 26 e e 39 ore[10]. Ciò nondimeno, possiamo comunque osservare una notevole variabilità dei tempi di lettura di ciascuna iterazione legata ad ogni riferimento individuato.

[10] L'andamento che ho riscontrato in fase di sperimentazione è irregolare, con andamento tra 1 e 3 secondi, con picchi ad intervalli irregolari. Tuttavia, l'analisi della prestazione è stata effettuata soltanto prendendo il tempo di ogni singola iterazione.



Un modo per affrontare questo problema è stato quello di creare una copia di backup dei dati analizzati fino ad un punto (per poi riprendere l'operazione successivamente), il codice è stato adattato opportunamente in modo tale che si possano selezionare il numero di iterazioni, oppure un tempo di esecuzione superati i quali il codice si fermi e salvi i risultati temporanei (creando una copia di backup dalla quale si possa riprendere le operazioni in un tempo successivo). Impostando l'esecuzione durante un paio di notti, è stato possibile eseguire tutta la mole di operazioni necessarie.

5.- **Descrizione dell'algoritmo.**

```
TT, ART, II = LETTURA(url0, n=None, m=None)
OUTPUT, ATTESA = CICLO_GENERALE(urlx, backup=, max=, TEMPUS=, reader=, xml_file=)
```

Il codice sviluppato contiene varie funzioni, quelle fondamentali sono la funzione LETTURA e CICLO_GENERALE (le quali aggregano opportunamente le altre) funzioni.
La Funzione LETTURA apre un collegamento URL (denominato in questo caso url0) ed estrae gli elementi:

- TT: Titolo dell'atto giuridico (se non si trova nessuna corrispondenza, ad esempio nel caso di un atto giuridico che non è presente nel sito normattiva.it, viene popolato con l'espressione TITOLO ASSENTE)
- ART: Numero dell'articolo e del comma (popolato solo nel caso in cui si tratti di una sotto-norma, altrimenti viene popolato con l'espressione SOTTOTITOLO ASSENTE)
- II: Una tabella in cui sono presenti due colonne che comprendono gli URL dei riferimenti giuridici individuati (ed unici) e gli URL_originali (ovverosia i riferimenti giuridici della norma generale a cui essi appartengono, informazione rilevante per studiare la relazione tra norme e sotto-norme)

Questa funzione può funzionare da sola, venendo utilizzata inserendo manualmente il collegamento URL. Oppure, è possibile far ripetere iterativamente queste operazioni in maniera che, una volta estratti i nuovi collegamenti, la medesima operazione possa essere svolta in automatico dal computer. Questa iterazione viene svolta all'interno della funzione CICLO_GENERALE[11].
Essa è costituita dai seguenti INPUT:

[11] All'interno della quale, vengono usati anche i parametri "n" ed "m" della funzione LETTURA. Questi parametri sono necessari per tenere conto del numero di cicli che si sta svolgendo e l'identificativo che era stato associato.

1) 1 variabile obbligatoria: urlx, che rappresenta l'indirizzo URL di normattiva.it associato alla norma giuridica che si desidera analizzare;
2) 5 variabili di INPUT facoltative: esse permettono una gestione migliore nel caso il codice impieghi notevole tempo per la esecuzione. Backup e xml_file rappresentano, rispettivamente, la cartella ed il nome (inclusa l'estensione .xml) del file XML che include i dataframe output ed attesa temporaneamente, max e TEMPUS rappresentano, rispettivamente, il numero massimo di cicli ed il numero massimo di secondi superati i quali la funzione salva i risultati nel documento XML, reader rappresenta una variabile binaria (0,1) tale per cui se è uguale ad 1 permette alla funzione di leggere il file effettuare il salvataggio temporaneo dei documenti nel documento XML di cui sopra

É importante sottolineare che il codice di cui sopra è stato strutturato in maniera tale che possa funzionare senza che necessiti di un intervento umano da quando inizi le operazioni di ricerca a quando termina. Si ritiene questa una condizione molto utile, in quanto permette di lasciare che il computer stesso possa svolgere tutte le operazioni di ricerca senza necessitare di altre interazioni con l'utente umano. L'output corrisponde a due tabelle (dataframe) aventi le stesse colonne, di cui:

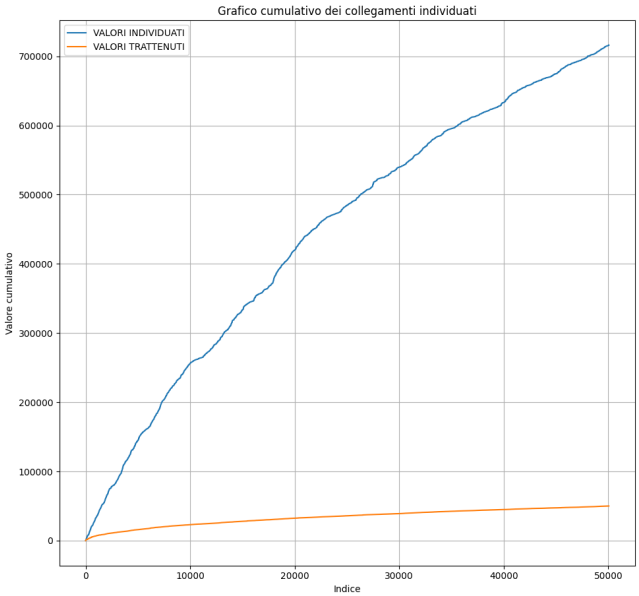
c) OUTPUT: Rappresenta la tabella dei risultati finali;
d) ATTESA: Rappresenta la tabella dei risultati che devono essere ancora analizzati.

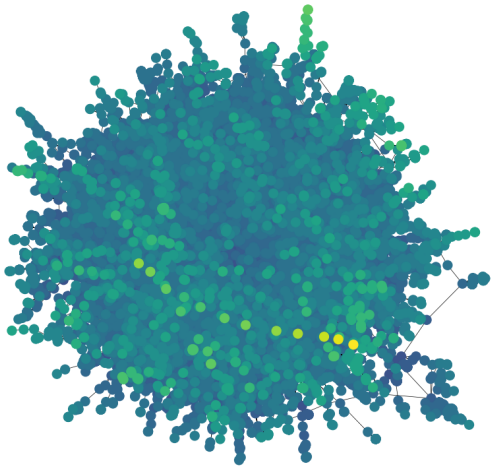
Se tutti i riferimenti sono stati analizzati, e la tabella OUTPUT sarà l'unica popolata, mentre la tabella ATTESA avrà un numero di righe pari a 0, vice versa, i dati che devono essere ancora letti sono registrati temporaneamente in ATTESA.

La descrizione delle operazioni è proposta in un diagramma di flusso illustrato in appendice. Infine, eseguendo i seguenti comandi:

```
grafo = build_graph_from_strings(OUTPUT['IDENTIFICATIVO'])
draw_graph(grafo)
```

Sarà possibile ottenere una rappresentazione del grafo ad albero associato alla norma giuridica iniziale (ciascun colore è associato al livello di profondità del nodo). Qui di seguito è riportato il grafo ad albero del seguente url:
https://www.normattiva.it/uri-res/N2Ls?urn:nir:presidente.repubblica:decreto:2013;70@originale





6.- **Possibili Sviluppi.**

Una metodologia del genere può essere integrata con altri tipi di analisi, come ad esempio l'analisi del testo sottostante (attraverso un'analisi del linguaggio naturale denominata N.L.P.). Anche la visualizzazione (realizzata attraverso matplotlib) è migliorabile. Si preferisce discutere in questa sede quali possano essere le possibili migliorie della metodologia applicata fino ad ora, per comprendere quanto si possa sfruttare, capire dove si possa arrivare, e soprattutto dove non si possa arrivare, pertanto considerando esclusivamente i seguenti passaggi:

1) Analisi dei Riferimenti espliciti;
2) Lettura dei collegamenti ipertestuali;

Il contributo principale di questo codice, così come è stato sviluppato fino ad ora, è il seguente:

É in grado di individuare tutti e soli i riferimenti giuridici presenti a partire da una norma iniziale (esaustività) ed è in grado di stabilire una relazione gerarchica tra di essi.

A questo punto, il passo successivo è rappresentato dalla sintesi testuale di questo corpus giuridico. Una possibile prima implementazione sarebbe la semplice sostituzione dei nodi con il testo sottostante. In tal modo, sarebbe possibile generare un testo dinamico, in formato H.T.M.L., che, al posto di ciascun riferimento esplicito sia sostituita da una frase del genere:

.. ai sensi della norma XXXX, la quale prevede che "YYYY"

In cui l'espressione "YYYY" corrisponderà al testo giuridico di riferimento.

Questo tipo di testo potrebbe non rappresentare il riassunto più efficace, per il semplice fatto che la gerarchia individuata dal grafo ad albero fino ad ora sviluppato non necessariamente rappresenta una gerarchia effettiva nell'importanza delle norme giuridiche e non necessariamente il modo migliore di riassumere un testo.

Già in fase di lettura dei dati, infatti, abbiamo a disposizione i riferimenti trovati e i riferimenti selezionati per l'elaborazione del grafo. Un riferimento giuridico, infatti, può apparire più volte nel corso della elaborazione della analisi stessa, non a caso la colonna OUTPUT['TROVATI'] può assumere valori maggiori di OUTPUT['SELEZIONATI'].

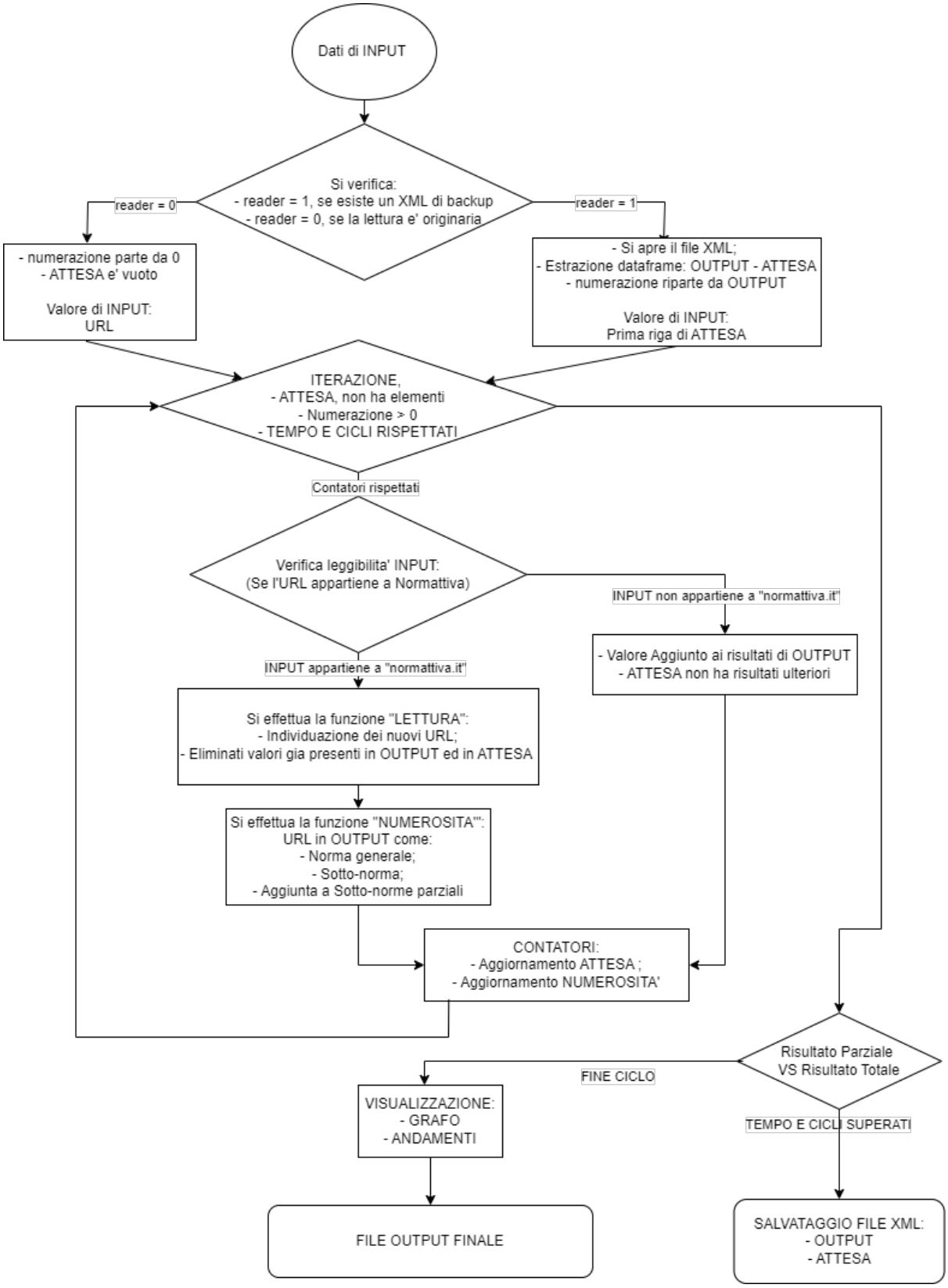
Il codice sviluppato fino ad ora non tiene conto dei riferimenti giuridici individuati successivamente poiché vengono eliminati dal risultato finale (l'unica informazione che si è ritenuta opportuna da registrare è stata la numerosità degli elementi eliminati). Da un punto di vista puramente tecnico, è facile includere questa informazione nell'output: basterebbe aggiungere una colonna che includesse, in ciascun suo elemento, la lista di URL eliminati.[12]

L'inclusione di questa ulteriore informazione, tuttavia, non dovrebbe essere fine a sé stessa, ma dovrebbe servire per la creazione di corde, collegamenti tra nodi non sequenziali all'interno del grafico.

[12] Ad esempio modificando la funzione FILTRO, presente all'interno della funzione CICLO_GENERALE, e modificando
Y4 = set(URL(Y3)) # SI PRENDONO LE OCCORRENZE UNICHE
All'interno della funzione LETTURA

7.- **Appendice.**

Diagramma di flusso



Funzione Python:
Il codice è stato stampato in varie caselle di testo per mantere l'indentazione[13]. In primo luogo è necessario caricare i pacchetti seguenti

```
# LISTA DI PACCHETTI DA INSTALLARE
import requests
import pandas as pd
import time
import matplotlib.pyplot as plt
import networkx as nx
from bs4 import BeautifulSoup
import re
import getpass
import os
import xml.etree.ElementTree as ET
```

Successivamente le funzioni di cui sotto, sarà possibile far funzionare il presente codice. Questa prima lista è necessaria per far funzionare correttamente la funzione LETTURA.

1) OVERTURE, apre il sito di Normattiva

```
def OVERTURE(url):
    try:
        response = requests.get(url)  # Usa la variabile url invece di x
        lista = []
        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            return soup
    except Exception as e:
        print(f"Errore durante l'apertura dell'URL {url}: {e}")
        soup = 'TESTO ASSENTE'
        return soup  # Restituisci None in caso di errore
```

2) TITOLO, se lo trova, legge il titolo dell'atto giuridico sottostante

```
# Identifica il Titolo dell'Atto
def TITOLO(soup):
    try:
        # Prova a trovare il titolo
        titolo_atto_div = soup.find('div', id='titoloAtto', class_='data_info text-center my-1')
        # Se trova l'elemento, estrai il testo
        if titolo_atto_div and titolo_atto_div.h2:
            titolo_atto = titolo_atto_div.h2.get_text(strip=True)
            # Normalizza il testo rimuovendo spazi extra
            titolo_atto = ' '.join(titolo_atto.split())
        else:
            # Se l'elemento non ha il titolo atteso, restituisce "TITOLO ASSENTE"
            titolo_atto = 'TITOLO ASSENTE'
    except Exception:
        # Se c'è un'eccezione durante il tentativo di trovare l'elemento, restituisce "TITOLO ASSENTE"
        titolo_atto = 'TITOLO ASSENTE'
    return titolo_atto
```

3) ARTICOLO, se esiste, legge il numero dell'articolo (e dell'eventuale comma) dell'atto giuridico sottostante

```
# Identifica il numero dell'articolo e del comma, se esistono
def ARTICOLO(url):
    # Verifica se la tilde (~) esiste nell'URL
    if '~' not in url:
        return 'SOTTOTITOLO ASSENTE'
    # Estrai la stringa dopo la tilde (~)
    after_tilde = url.split('~', 1)[1]
    # Esegui le sostituzioni richieste
    after_tilde = after_tilde.replace("art", "ARTICOLO ")
    after_tilde = after_tilde.replace("-", " - ")
    after_tilde = after_tilde.replace("com", "COMMA ")
    return after_tilde
```

4) TESTO, se esiste, estrae il testo dell'atto giuridico sottostante

```
# Estrae il Testo Sottostante
def TESTO(soup):
    try:
        body_testo = soup.find('div', class_='bodyTesto')
        if body_testo:
            return str(body_testo)
        else:
            print("Nessun nodo <div class='bodyTesto'> trovato.")
            soup = 'CORPO DEL TESTO ASSENTE'
            return soup
    except Exception as e:
        print(f"Errore durante l'analisi del testo HTML: {e}")
        soup = 'ANALISI CORPO DEL TESTO FALLITA'
        return soup
```

5) RIMOZIONE, si attiverà se la funzione TESTO ha dato un output valido:

[13] La dimensione del carattere di questi codici è stato volutamente piccolo per permettere che l'indentazione e gli eventuali commenti possano rimanere intatti.

```
def RIMOZIONE(soup):
    try:
        # Converti il testo HTML in un oggetto BeautifulSoup
        soup = BeautifulSoup(soup, 'html.parser')
        # Trova tutti i nodi con classi che contengono "preamble" o "note-akn"
        nodes_to_remove = soup.find_all(class_=lambda c: c and ('preamble' in c or 'note-akn' in c))
        # Rimuovi i nodi trovati
        for node in nodes_to_remove:
            node.extract()
        # Restituisci il nuovo oggetto BeautifulSoup
        return soup
    except Exception as e:
        print(f"Errore durante la rimozione dei nodi di preamble e note-akn: {e}")
        return None
```

6) URL, estrae i collegamenti ipertestuali (dal testo individuato dalla funzione TESTO e RIMOZIONE):

```
def URL(soup):
    try:
        links = soup.find_all('a', href=True)
        # Estrai gli URL dalla lista di tag 'a' e aggiungi il prefisso se necessario
        urls = [link['href'] if link['href'].startswith('http') else "https://www.normattiva.it" + link['href'] for link in links]
        return urls
    except Exception as e:
        print(f"Errore durante l'estrazione degli URL dal testo HTML: {e}")
        return None
```

7) modifica, uniforma gli URL di normattiva in modo tale che si eliminano le versioni di altri periodi (e si utilizzino solo le versioni originali):

```
def modifica(lista_url):
    url_modificati = []
    for url in lista_url:
        if 'normattiva.it' in url:
            # Cerca "!vig=" seguito da una data (yyyy-mm-dd) oppure solo "!vig="
            url_modificato = re.sub(r'!vig=(\d{4}-\d{2}-\d{2}|)', '', url)
            url_modificato = re.sub(r'@originale', '', url_modificato)
            # Sostituisci "@originale~" con "~"
            url_modificato = re.sub(r'@originale~', '~', url_modificato)
            # Aggiungi "@originale" alla fine se non presente e se non c'è il tilde "~"
            if not url_modificato.endswith('@originale') and '~' not in url_modificato:
                url_modificato += '@originale'
        else:
            url_modificato = url
        url_modificati.append(url_modificato)
    return url_modificati
```

8) LISTA_ORIGINALE, estrae la versione originale della norma generale a cui un URL di Normattiva appartiene (rilevante se l'URL di INPUT è una sotto-norma):

```
def LISTA_ORIGINALE(lista_url):
    url_modificati = []
    for url in lista_url:
        if 'normattiva.it' not in url:  # Gli URL non di Normattiva equivalgono al proprio originale
            url_modificati.append(url)
        else:
            # Elimina tutto quello che si trova dopo "~" oppure dopo "!vig="
            if '~' in url:
                url = url.split('~', 1)[0]
            if '!vig=' in url:
                url = url.split('!vig=', 1)[0]
            # Se, nel nuovo URL non compare alla fine "@originale" allora aggiungilo
            if not url.endswith('@originale'):
                url += '@originale'
            url_modificati.append(url)
    return url_modificati
```

A questo punto si può introdurre la funzione LETTURA; essa individua tutti i collegamenti ipertestuali di norme valide a partire da un url di input. É stata strutturata in maniera tale per cui possa essere utilizzata sia da sola, sia all'interno del codice ad un punto successivo. Nel caso si utilizzasse da sola, è sufficiente inserire come parametro l'url di input url0, mentre non sarà necessario utilizzare i parametri n ed m (utili invece successivamente). Il suo output è costituito da 3 elementi: due stringhe di testo ed una tabella (dataframe).

```
def LETTURA(url0, n=None, m=None):
    Y1 = OVERTURE(url0)
    # Se l'URL non si apre, allora l'Output sarà una tabella vuota
    if not Y1:
        Singola_pagina = pd.DataFrame(columns=['URL', 'NOMI', 'COMPONENTI', 'Tempi'])  # Creare un DataFrame vuoto
    else:
        # Possiamo a questo punto stampare il nominativo dell'atto giuridico che si sta analizzando e gestire il caso in cui non abbia un nome
        # Se "TITOLO" o "ARTICOLO" sono assenti, essi daranno come risposta un valore vuoto
        T1 = TITOLO(Y1)
        ART = ARTICOLO(url0)
        if T1 and ART:
            print(f"CICLO {n} di {m} - IN ESECUZIONE {T1} - {ART}")
        if T1 and not ART:
            print(f"CICLO {n} di {m} -IN ESECUZIONE {T1}")
            ART = 'SOTTOTITOLO ASSENTE'
        if not T1 and not ART:
            print(f"CICLO {n} di {m} -IN ESECUZIONE ATTO SENZA NOMINATIVO")
            ART = 'SOTTOTITOLO ASSENTE'
            T1 = 'TITOLO ASSENTE'
        if not T1 and ART:
            print(f"CICLO {n} di {m} -IN ESECUZIONE ATTO SENZA TITOLO")
            T1 = 'TITOLO ASSENTE'
        ###
        ###
        Y2 = TESTO(Y1) ### A QUESTO PUNTO POSSIAMO APRIRE IL TESTO E RICERCARE
        # GESTIAMO IL CASO IN CUI QUESTO ALGORITMO NON DIA UN RISULTATO
        if not Y2:
            Singola_pagina = pd.DataFrame(columns=
                              ['controllo_URL',
                               'URL_originale',
                               'URL',
                               'NOMI',
                               'COMPONENTI',
                               'IDENTIFICATIVO',
                               'TEMPO',
                               'TROVATI',
                               'SELEZIONATI'])
        # GESTIAMO IL CASO IN CUI VADA TUTTO BENE
        else:
            Y3 = RIMOZIONE(Y2)
            Y4 = set(URL(Y3)) # SI PRENDONO LE OCCORRENZE UNICHE
            Y4_list = list(Y4)  # Trasformare l'insieme in una lista
            Singola_pagina = pd.DataFrame({
                               'URL_originale': LISTA_ORIGINALE(Y4_list),
                              'URL': modifica(Y4_list), 'NOMI': '',
                              'COMPONENTI': '',
                              'IDENTIFICATIVO': '',
                              'TEMPO': '',
                              'TROVATI': '',
                              'SELEZIONATI': ''})
    return T1, ART, Singola_pagina
```

Le funzioni seguenti sono necessarie per far funzionare correttamente la funzione denominata CICLO_GENERALE.

Le 3 funzioni di seguito gestiscono il processo di creazione di un file XML temporaneo (nel caso si desiderasse far interrompere l'esecuzione della funzione finale prima che essa si sia conclusa, dopo un numero N di cicli, oppure dopo un numero S di secondi di esecuzione)

1) AGGIUSTA_NOME_DOCUMENTO, aggiusta il nome estratto dalla norma giuridica, per utilizzarlo (se necessario) come nome del file di salvataggio temporaneo:

```
def AGGIUSTA_NOME_DOCUMENTO(filename):
    # Caratteri non consentiti nei nomi dei file
    illegal_chars = ['<', '>', ':', '"', '/', '\\', '|', '?', '*', ' ']
    # Sostituzione dei caratteri non consentiti con un carattere consentito
    for char in illegal_chars:
        filename = filename.replace(char, '_')
    # Rimozione degli spazi bianchi all'inizio e alla fine del nome del file
    filename = filename.strip()
    # Aggiunta dell'estensione .xml
    filename += '.xml'
    return filename
```

2) SALVA_XML_TEMPORANEO: salva i dataframe OUTPUT ed ATTESA nella cartella di salvataggio temporaneo;

```
def SALVA_XML_TEMPORANEO(df1, output_folder, output_file, df2=None):
    # Crea il percorso completo del file XML di destinazione
    output_path = os.path.join(output_folder, output_file)
    # Crea un elemento radice per il file XML
    root = ET.Element('data')
    # Funzione per aggiungere i dati di un DataFrame come sotto-elementi
    def add_dataframe_to_xml(df, parent_element, dataframe_name):
        dataframe_element = ET.SubElement(parent_element, dataframe_name)
        for index, row in df.iterrows():
            record_element = ET.SubElement(dataframe_element, 'record')
            for col in df.columns:
                value = str(row[col])
                ET.SubElement(record_element, col).text = value
    # Aggiungi df1 come elemento principale
    add_dataframe_to_xml(df1, root, 'OUTPUT')
    # Se df2 è fornito, aggiungilo al file XML
    if df2 is not None:
        add_dataframe_to_xml(df2, root, 'ATTESA')
    # Crea un albero XML e scrivi il contenuto su un file
    tree = ET.ElementTree(root)
    tree.write(output_path, encoding='utf-8', xml_declaration=True)
```

3) LEGGI_XML_TEMPORANEO: apre i dataframe OUTPUT ed ATTESA precedentemente salvati;

```
def LEGGI_XML_TEMPORANEO(xml_folder, xml_file):
    # Parsing del file XML
    xml_path = os.path.join(xml_folder, xml_file).replace('\\', '//')
    xml_path = xml_path + ".xml"
    tree = ET.parse(xml_path)
    root = tree.getroot()
    # Estrazione dei dati dal nodo 'OUTPUT'
    data_output = []
    for record_element in root.findall('./OUTPUT/record'):
        record = {}
        for child in record_element:
            record[child.tag] = child.text
        data_output.append(record)
    # Estrazione dei dati dal nodo 'ATTESA' (se presente)
    data_attesa = []
    # Controllo per evitare errore se 'ATTESA' non è presente
    if root.find('./ATTESA') is not None:
        for record_element in root.findall('./ATTESA/record'):
            record = {}
            for child in record_element:
                record[child.tag] = child.text
            data_attesa.append(record)
    # Creazione dei DataFrame
    df_output = pd.DataFrame(data_output)
    df_attesa = pd.DataFrame(data_attesa)  # Questo sarà vuoto se 'ATTESA' non è presente
    return df_output, df_attesa
```

Le seguenti tre funzioni sono necessarie per distinguere i riferimenti a norme in siti differenti da Normattiva, e per gestire la contemporanea presenza di norme e sottonorme all'interno del ciclo di letture.

1) CONTROLLO_ARTICOLO: controlla se, dato un INPUT una sotto-norma, sia presente nella lista dei riferimenti individuati fino ad ora la norma generale associata;

```
def CONTROLLO_ARTICOLO(url, OUTPUT, ATTESA):
    result = 1
    if '~' in url and 'normattiva.it' in url:
        ORIGINALE = LISTA_ORIGINALE([url])[0]
        url_originale_totale = pd.concat([OUTPUT[['URL_originale', 'URL']], ATTESA[['URL_originale', 'URL']]], ignore_index=True)
        contains_no_tilde = url_originale_totale[url_originale_totale['URL_originale'] == ORIGINALE]['URL'].str.contains('~').any() == False
        if contains_no_tilde:
            result = 0  # Se esiste almeno un elemento che non contiene tilde
        else:
            result = 1  # Se tutti gli elementi contengono tilde
    return result
```

2) NUMERO_ARTICOLO: assegnazione dell'identificativo, che tiene conto se la norma è una sotto-norma o meno, se non appartiene a Normattiva o se si registrano norme e sotto-norme nella lista dei risultati;

```
def NUMERO_ARTICOLO(url, OUTPUT, ATTESA):
    # VARIE CONDIZIONI
    if '~' not in url and 'normattiva.it' not in url:                           # 1. URL al di fuori di Normattiva -> Allora assente articolo e comma
        return '(E)'
    if '~' in url and 'normattiva.it' in url:                   # 2. URL di Normattiva con Articolo e comma: Inserisci articolo e comma
        # Estrai la stringa dopo la tilde (~)
        after_tilde = url.split('~', 1)[1]
        # Verifica se "!vig=" è presente nell'URL
        if '!vig=' in after_tilde:
            # Elimina cio' che e' presente dopo "!vig=" e la stringa "@originale"
            after_vig = after_tilde.split('!vig=', 1)[0].split('@originale', 1)[0]
        else:
            # Se "!vig=" non è presente, utilizza l'intera stringa dopo la tilde
            after_vig = after_tilde
            # Elimina l'espressione "art" e sostituisce "-com" (oppure solo "com" se "-" non è presente) con "_"
            after_vig = after_vig.replace("-", "")
            after_vig = after_vig.replace("art", "").replace("com", "_")
        return f"({after_vig})"
    if '~' not in url and 'normattiva.it' in url:                               # 3. URL di Normattiva Generale: Inserisci articolo e comma
        ORIGINALE = LISTA_ORIGINALE([url])[0]                   # 3.i - Si trova l'URL Originale associato
        url_originale_totale = pd.concat(
                               [OUTPUT[['URL_originale', 'URL']],
                               ATTESA[['URL_originale', 'URL']]],
                               ignore_index=True)# 3.ii - Si genera l'Unione degli URL originali finora trovati
        url_originale_totale = url_originale_totale[
                                url_originale_totale['URL'].str.contains('~') &
                                url_originale_totale['URL'].str.contains('normattiva.it')]
        url_originale_totale = url_originale_totale[url_originale_totale['URL_originale']==ORIGINALE]['URL_originale']
        if url_originale_totale.shape[0] > 0:
            return '(C)'                        # 3.iii - Se esiste corrispondenza, allora sara' un riferimento complementare
        else:
            return '(T)'                        # 3.iv - Altrimenti una nuova norma generale
```

3) FILTRO: Ad ogni iterazione, elimina i nuovi risultati che sono giá stati individuati;

```
def FILTRO(I, O, A):
    ####
    #### 1. INPUT
    ####
    RIGA = pd.concat([O['URL'], A['URL']], axis=0).reset_index(drop=True)       # 1.i - Unisce gli URL già trovati
    N_art = RIGA[RIGA.str.contains("normattiva.it") &
                                ~RIGA.str.contains("~")]                        # 1.ii - Trova gli URL già trovati che non siano sotto-norme in normattiva
    I1 = I[['URL', 'URL_originale']]                                            # 1.iii - Importa gli URL e i loro riferimenti originali da analizzare
    # 1.iv - Gli input sono divisi in due tabelle
    I_art = I[I['URL'].str.contains("normattiva.it") &
                                I['URL'].str.contains("~")]  # 1.iv.a - URL da analizzare che siano sotto-norme in normattiva
    I_oth = I1[~I1['URL'].isin(I_art['URL'])]                   # 1.iv.b - URL da analizzare che non siano sotto-norme in normattiva
    ####
    #### 2. OPERAZIONI
    ####
    I_art = I_art[~I_art['URL'].isin(RIGA)]     # 2.i.a - Elimina gli URL già trovati
    I_oth = I_oth[~I_oth['URL'].isin(RIGA)]     # 2.i.b - Elimina gli URL già trovati
    I_oth = I_oth[~I_oth['URL_originale'].isin(N_art)]          # 2.ii - Elimina gli URL di sottonorme, le cui norme generali sono già state trovate
    OUTPUT = pd.concat([I_art['URL'], I_oth['URL']],
            axis=0).reset_index(drop=True)                      # 2.iii.a - Lista di URL di OUTPUT
    OUTPUT = I[I['URL'].isin(OUTPUT)]                           # 2.iii.b - Creazione OUTPUT
    return OUTPUT
```

A questo punto la funzione CICLO_GENERALE aggrega tutte le funzioni di cui sopra. Dato come input l'URL di una norma che si desidera analizzare, questa funzione può procedere nella lettura di tutti i riferimenti giuridici che siano legati ad essa, assegnando a ciascuno di essi un identificativo unico, il quale tiene conto della posizione relativa la norma iniziale su un grafo ad albero. Nel caso la variabile reader sia diversa da 0, sarà necessario specificare anche le altre variabili opzionali.

```
def CICLO_GENERALE(url, backup=None, max=None, TEMPUS=None, reader=0, xml_file=None):
    TT = time.time()
    TTMAX = 0
    if reader == 1:
        OUTPUT, ATTESA = LEGGI_XML_TEMPORANEO(backup, xml_file)
        n = OUTPUT.shape[0]
        print("-----")
        print(f"FILE APERTO, SI RI-COMINCIA DAL CICLO {n}")
        print("-----")
        max = max + n
    else:
        n = 0 # E' il numero di cicli
        print("-----")
        print(f"FILE APERTO, CICLO ORIGINARIO")
        print("-----")
        ATTESA = pd.DataFrame(columns=['URL_originale','URL', 'NOMI', 'COMPONENTI', 'IDENTIFICATIVO','TEMPO','TROVATI','SELEZIONATI'])
    while (ATTESA.shape[0] > 0 and (not max or n <= max) and (not TEMPUS or TTMAX <= TEMPUS)) or (ATTESA.shape[0] == 0 and n == 0):
    # Il tuo codice qui
        if n == 0:
            tt = time.time()
            T1, ART, II = LETTURA(url, 0, 0)
            tt = time.time() - tt
            OUTPUT = pd.DataFrame({'URL_originale': [modifica([url])],
                                   'URL': [url],
                                   'NOMI': [T1 if T1 else ''],
                                   'COMPONENTI': [ART if ART else ''],
                                   'IDENTIFICATIVO': ['X'],
                                   'TEMPO': [round(tt, 1)],
                                   'TROVATI': [II.shape[0]],
                                   'SELEZIONATI': [II.shape[0]]})
            ATTESA = pd.concat([ATTESA, II], ignore_index=True)
            List = (II['URL']).reset_index(drop=True)
            ATTESA['IDENTIFICATIVO'] = [f'X.{i}{NUMERO_ARTICOLO(List[i-1], OUTPUT, ATTESA)}' for i in range(1, len(ATTESA) + 1)]
        # Gestiamo le Iterazioni successive
        if n > 0:
            ## Viene eseguito il ciclo nei valori in ATTESA
            tt = time.time()
            uu = ATTESA['URL_originale']
            ll = ATTESA['URL']
            ii = ATTESA['IDENTIFICATIVO']
            mm = OUTPUT['URL']
            if 'www.normattiva.it' not in ll[0]:
                print(f"CICLO {n} di {ii[0]} NON HA DATO RISULTATI PERCHE' NON APPARTIENE A NORMATTIVA.IT - {ll[0]}")
                # Altrimenti, assegna valori di default
                T1 = 'Titolo Assente'
                ART = 'Sottotitolo Assente'
                II = pd.DataFrame(columns=['URL_originale','URL', 'NOMI', 'COMPONENTI', 'IDENTIFICATIVO', 'TEMPO','TROVATI','SELEZIONATI'])
                new_row = pd.DataFrame({'URL_originale': 'NO_ORIGINALE',
                                        'URL': ll[0],
                                        'NOMI': T1,
                                        'COMPONENTI': ART,
                                        'IDENTIFICATIVO': ii[0],
                                        'TEMPO': 0,
                                        'TROVATI': 0,
                                        'SELEZIONATI': 0}, index=[0])
                OUTPUT = pd.concat([OUTPUT, new_row], ignore_index=True)
            else:
                XXX = CONTROLLO_ARTICOLO(url, OUTPUT, ATTESA)
                if XXX == 0:
                    print(f"CICLO NUMERO {n} --- ELEMENTO ll[0] ELIMINATO PERCHE' GIA' PRESENTE LA NORMA GENERALE")
                else:
                    T1, ART, II = LETTURA(ll[0], n, ATTESA['IDENTIFICATIVO'][0])
                    trovati = II.shape[0]
                    tt = time.time() - tt
                    if II.shape[0] == 0:
                        print(f"QUESTA NORMA NON HA ULTERIORI RIFERIMENTI")
                    if II.shape[0] > 0:
                        lunghezza = II.shape[0]
                        # II = filter_and_merge_dataframes(OUTPUT, ATTESA, II)
                        II = FILTRO(II, OUTPUT, ATTESA)
                        lunghezza1 = II.shape[0]
                        if lunghezza > lunghezza1:
                            print(f"ELIMINATI {lunghezza - lunghezza1} VALORI DA {lunghezza}")
                        else:
                            print(f"NESSUN VALORE ELIMINATO, I VALORI SONO E RESTANO DI {lunghezza} ELEMENTI")
                        if II.shape[0] > 0:
                            List = (II['URL']).reset_index(drop=True)
                            II['IDENTIFICATIVO'] = [f"{ii[0]}.{i+1}{NUMERO_ARTICOLO(List[i], OUTPUT, ATTESA)}" for i in range(II.shape[0])]
                            ATTESA = pd.concat([ATTESA, II], ignore_index=True)
                    new_row = pd.DataFrame({'URL_originale': uu[0],
                                            'URL': ll[0],
                                            'NOMI': [T1 if T1 else ''],
                                            'COMPONENTI': [ART if ART else ''],
                                            'IDENTIFICATIVO': ii[0],
                                            'TEMPO': [round(tt, 1)],
                                            'TROVATI': trovati,
                                            'SELEZIONATI': II.shape[0]})
                    OUTPUT = pd.concat([OUTPUT, new_row], ignore_index=True)
            #### Si elimina dai valori di ATTESA la norma appena analizzata
            ATTESA = ATTESA.drop(ATTESA.index[0]).reset_index(drop=True)
            #### Si aggiorna il numero di ciclo
        n = n + 1
        TTMAX = time.time() - TT
    if backup is None:
        backup = 'C://Users//' + getpass.getuser() + '//Documents'
    if xml_file is None:
        xml_file = OUTPUT['NOMI'][0]
    if ATTESA.shape[0] > 0:
        SALVA_XML_TEMPORANEO(OUTPUT, backup, AGGIUSTA_NOME_DOCUMENTO(xml_file), ATTESA)
        print("-----")
        print(f"FILE DI RECOVERY SALVATO, CI SONO {OUTPUT.shape[0]} ELEMENTI ANALIZZATI ED {ATTESA.shape[0]} ELEMENTI DA ANALIZZARE RESIDUALI, CHIAMATO {xml_file} E SALVATO {backup}")
        print("-----")
    if ATTESA.shape[0] == 0:
        SALVA_XML_TEMPORANEO(OUTPUT, backup, AGGIUSTA_NOME_DOCUMENTO(xml_file))
        print("-----")
        print(f"CICLO TERMINATO, ESISTONO {OUTPUT.shape[0]} ELEMENTI ANALIZZATI --- IL FILE E' STATO CHIAMATO {xml_file} E SALVATO {backup}")
        print("-----")
    return OUTPUT, ATTESA
```

Che può essere attivato attraverso questa stringa di codice:

```
cartellina = 'C://Users//....
urlx = "https://www.normattiva.it/uri-res/N2Ls?urn:nir:presidente.repubblica:decreto:2013;70@originale"
OUTPUT, ATTESA = CICLO_GENERALE(urlx, backup=cartellina, max=1000,   TEMPUS=900, reader=1, xml_file='ESPERIMENTO')
```

Per la visualizzazione:

```
#### INIZIO VISUALIZZAZIONE UNICA
import matplotlib.pyplot as plt
import networkx as nx

def GRAFICO_SERIE_CUMULATA(dataframe, column1, column2, ax):
    # Estrai le serie dal DataFrame
    dataframe[column1] = pd.to_numeric(dataframe[column1], errors='coerce')  # Ora è numerico
    dataframe[column2] = pd.to_numeric(dataframe[column2], errors='coerce')  # Anche questo è numerico
    serie1 = dataframe[column1].cumsum()  # Calcola l'andamento cumulativo della prima serie
    serie2 = dataframe[column2].cumsum()  # Calcola l'andamento cumulativo della seconda serie
    # Plotta nel subplot assegnato
    ax.plot(serie1, label='VALORI INDIVIDUATI')  # Plotta l'andamento cumulativo della prima serie
    ax.plot(serie2, label='VALORI TRATTENUTI')  # Plotta l'andamento cumulativo della seconda serie
    ax.set_xlabel('Indice')  # Etichetta dell'asse x
    ax.set_ylabel('Valore cumulativo')  # Etichetta dell'asse y
    ax.set_title('Grafico cumulativo dei collegamenti individuati')  # Titolo del plot
    ax.legend()  # Mostra la legenda
    ax.grid(True)  # Mostra le linee della griglia

def GRAFICO_STRINGHE(string_list):
    G = nx.Graph()
    for string in string_list:
        nodes = string.split('.')
        parent = None
        level = 0
        for i, node in enumerate(nodes):
            if i == 0:
                parent = node
                G.add_node(parent, level=level)
            else:
                current_node = f"{parent}.{node}"
                level += 1
                G.add_node(current_node, level=level)
                G.add_edge(parent, current_node)
                parent = current_node
    return G

def DISEGNA_GRAFICO(G, ax):
    pos = nx.spring_layout(G)  # Layout algorithm
    levels = set(nx.get_node_attributes(G, 'level').values())
    cmap = plt.cm.get_cmap('viridis', len(levels))
    node_colors = [cmap(level) for level in nx.get_node_attributes(G, 'level').values()]
    # Impostazione personalizzata delle larghezze degli archi
    edge_widths = [0.5] * len(G.edges())
    # Disegna il grafo nel subplot assegnato
    nx.draw(G, pos, ax=ax, with_labels=False, node_size=100, node_color=node_colors, edge_color='black', width=edge_widths)

# Crea i subplot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))  # 1 riga, 2 colonne
# Utilizza la prima asse per il grafico cumulativo
GRAFICO_SERIE_CUMULATA(OUTPUT, 'TROVATI', 'SELEZIONATI', ax1)
# Costruisci il grafo e disegnalo nella seconda asse
grafo = GRAFICO_STRINGHE(OUTPUT['IDENTIFICATIVO'])
DISEGNA_GRAFICO(grafo, ax2)
# Mostra i subplot
plt.tight_layout()  # Migliora la disposizione del layout
plt.show()
```

Il cui codice restituirà come output il grafo ad albero e la tabella delle prestazioni includendo sia i riferimenti trovati, sia quelli selezionati. Per la visualizzazione dei tempi di esecuzione:

```
# Esempio di conversione dei valori in numeri
valori_numerici = pd.to_numeric(OUTPUT['TEMPO'], errors='coerce')

# Crea il grafico
plt.plot(valori_numerici)

# Aggiungi etichette e titolo
plt.xlabel('Indice')
plt.ylabel('Valore')
plt.title('Grafico dei Valori')

# Mostra il grafico
plt.show()
```

Il presente codice implementa un automa a pila. I parametri che di solito sono stati utilizzati sono stati, attraverso la calendarizzazione di esecuzione programmi:

- max = 30000 (cicli);
- Tempus = 36000 (tempo di esecuzione, 10 ore).

Abstract.- Si propone e si offre una spiegazione di un algoritmo, scritto in Python, per la ricerca esaustiva dei riferimenti giuridici di una norma dell'ordinamento italiano utilizzando la base dati del sito Normattiva.it. Il codice implementa un automa a pila e restituisce una rappresentazione del grafo dei riferimenti giuridici espliciti di una data norma giuridica.

It is provided and it is explained an algorithm, written in Python, for the exhaustive research of juridical reference from a law of the Italian Jurisdiction, using the Institutional database Normattiva.it. Such code implements a pushdown automaton and returns a tree chart representation of the explicit juridical reference starting from a given juridical rule.