Automatisation de la vérification des faits

BELKHIRI Taha

TIPE

Introduction

Sommaire

Généralitées sur les graphes de connaissances

- Le graphe de connaissances représente une collection de descriptions interconnectées d'entités
- Un graphe de connaissances est un ensemble de triplets (sujet, prédicat, objet).
- Le sujet et l'objet sont représentés par deux sommets différents, le prédicat représente l'arête qui les lie.

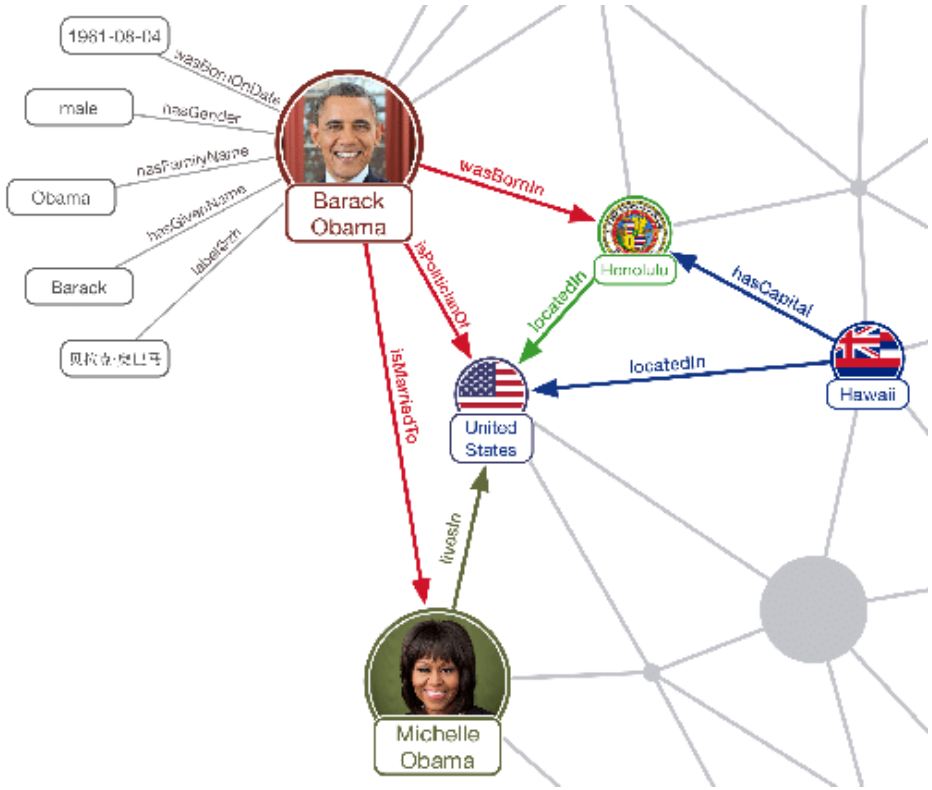


Figure 1: Représentation d'un graphe de connaissances

Construction d'un graphe de connaissance

a- À partir du dump Wikidata:

- Contient plus de 90.000.000 éléments
- 60Gb de données
- Contient des informations non pertinentes pour la construction du graphe de connaissances

b- Construction dynamique:

- Plus simple et plus facile à manier
- Ne demande pas autant d'espace
- Délocalisée

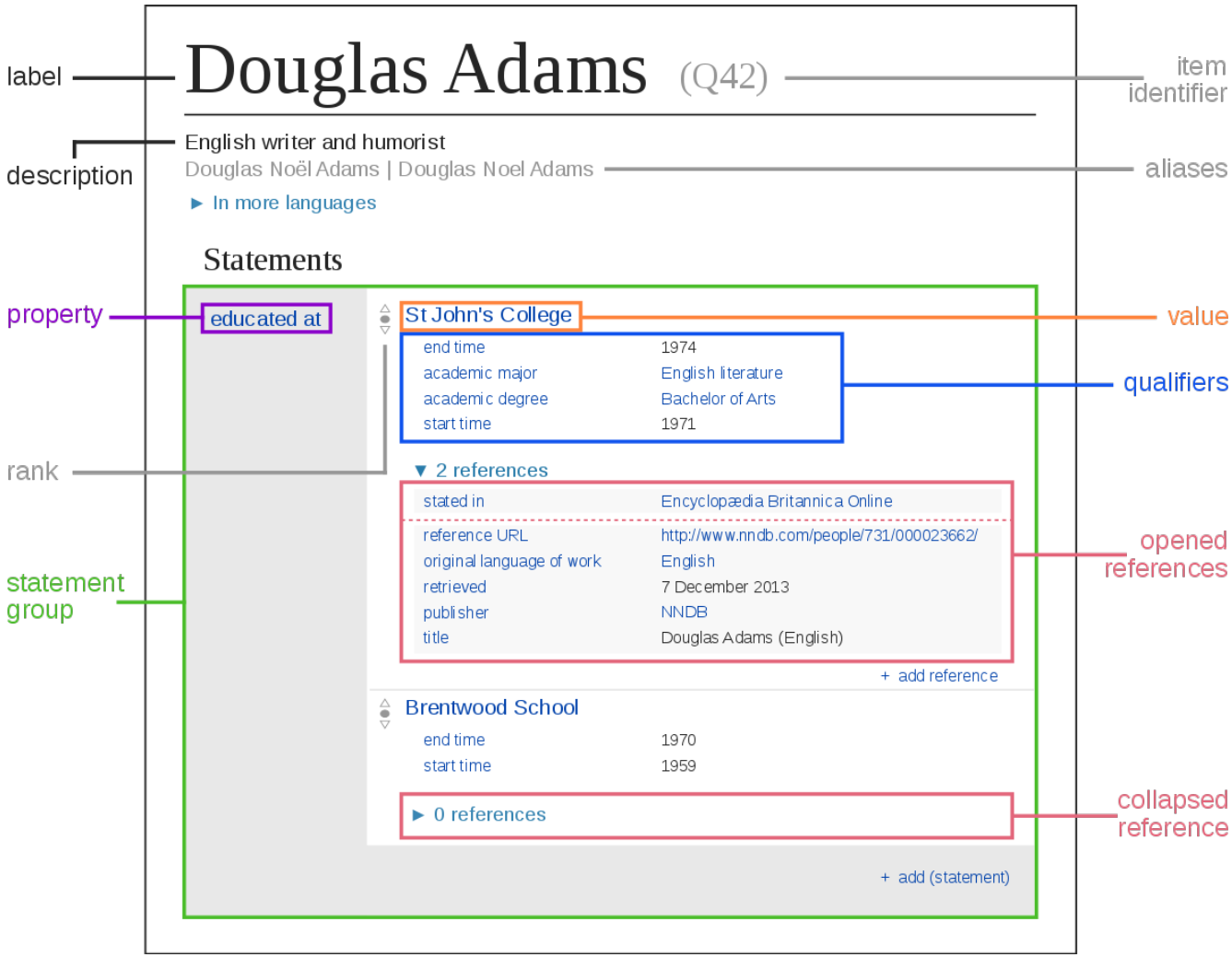


Figure 2: Modèle de données dans Wikidata

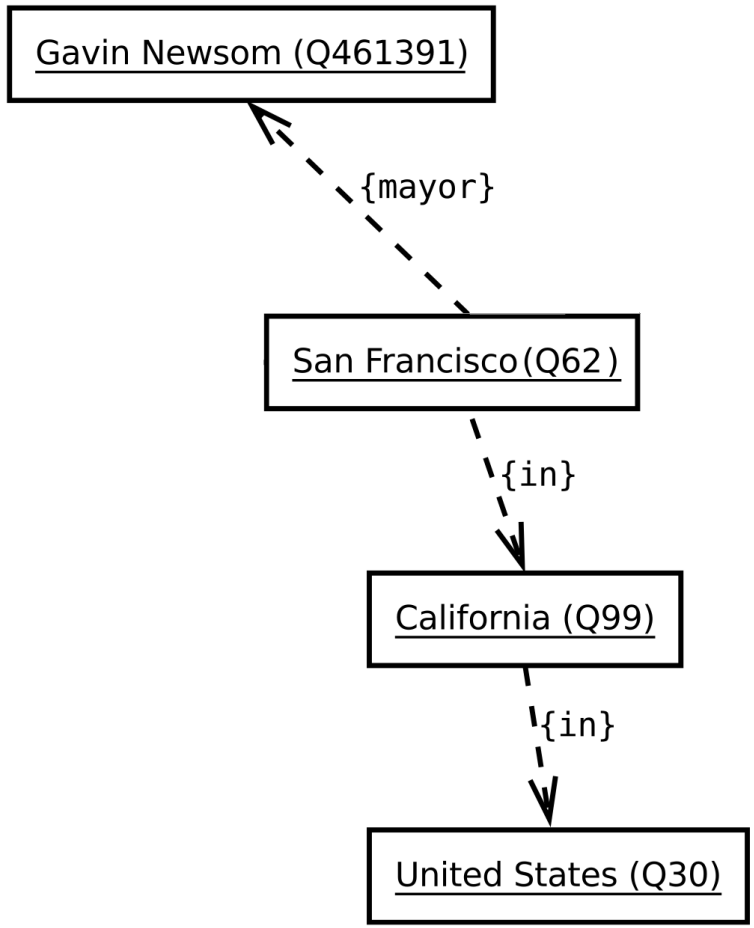


Figure 3: Construction d'une chaîne de données liées à partir de Wikidata

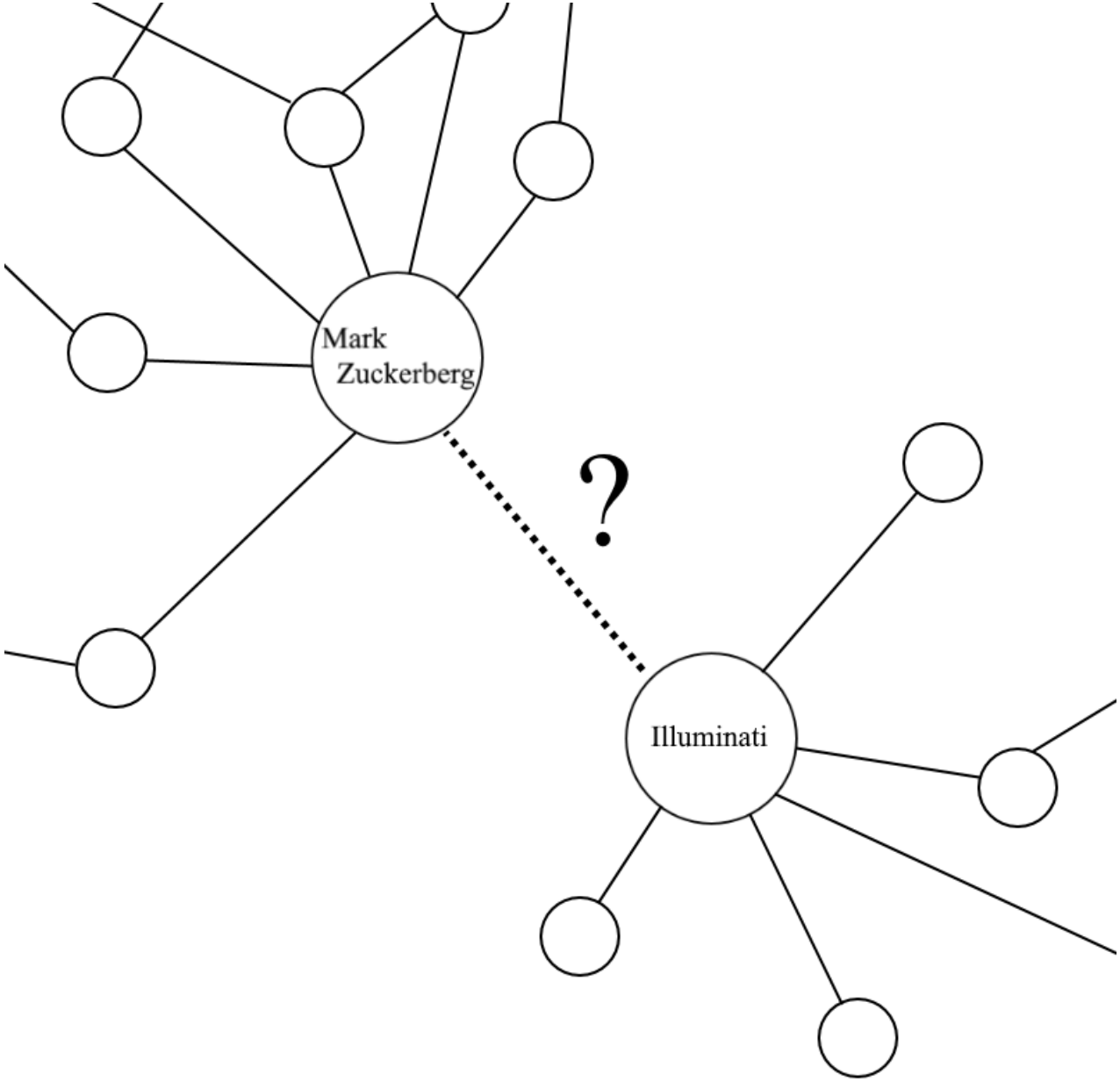


Figure 4: Recherche d'un chemin entre Mark Zuckerberg et les Illuminatis

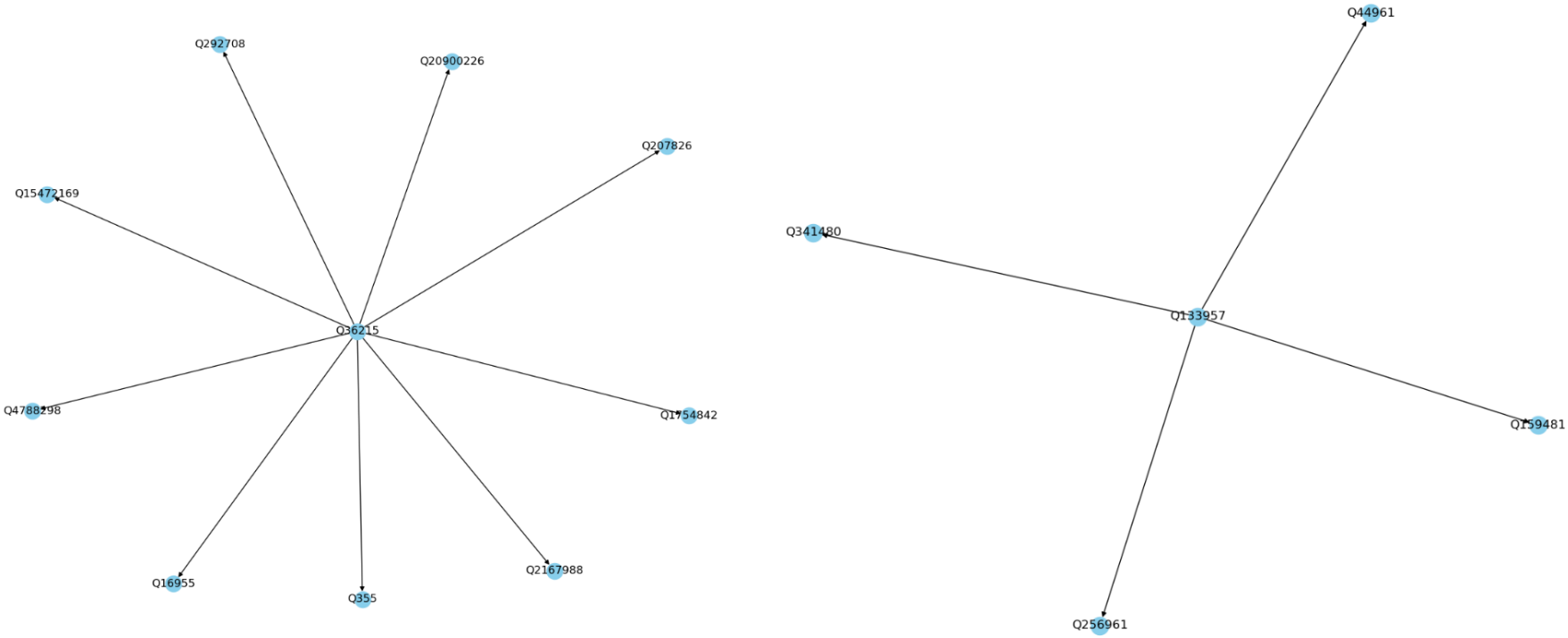


Figure 5: Première étape de la construction du graphe

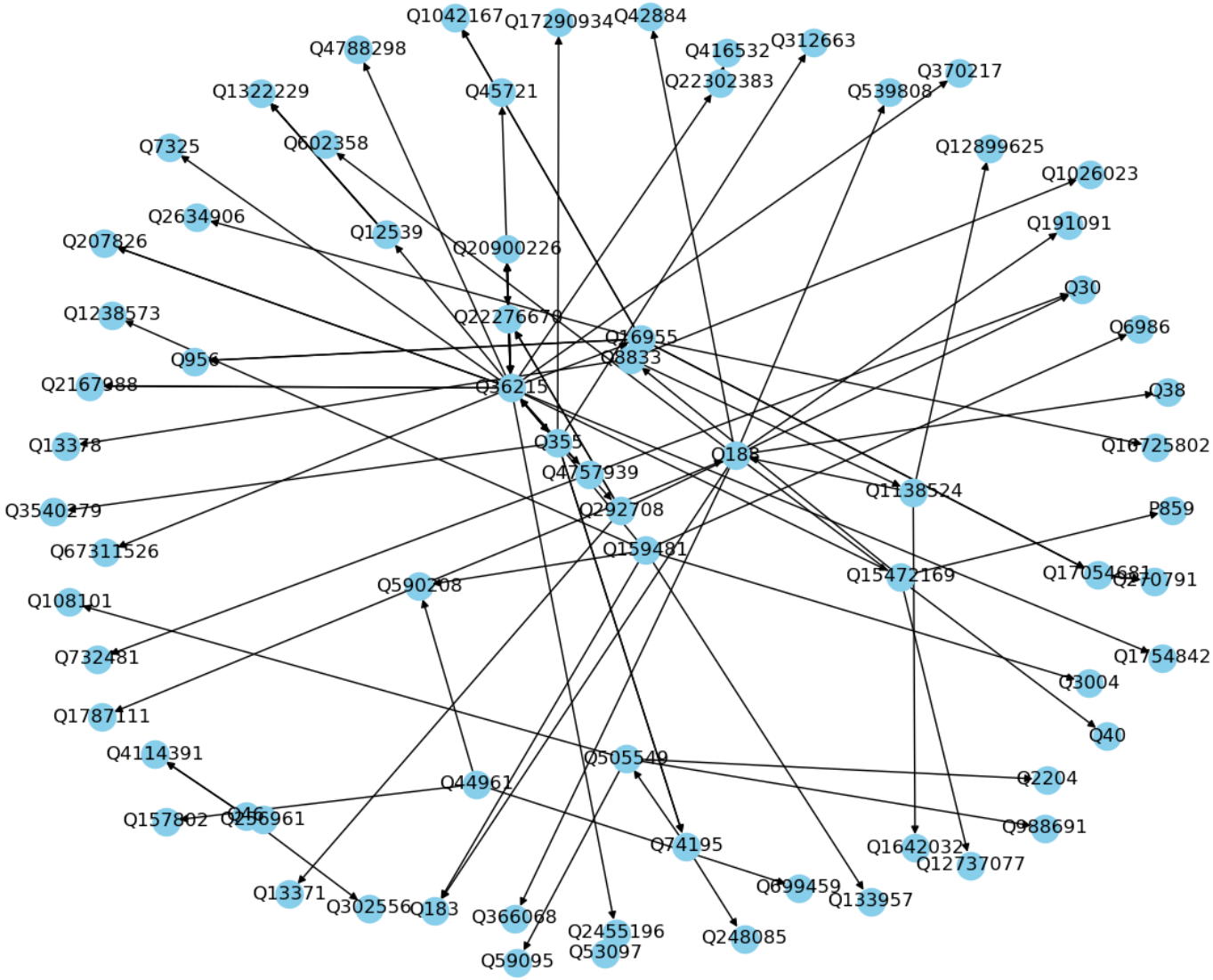


Figure 6: Graphe final

Valeur de vérité

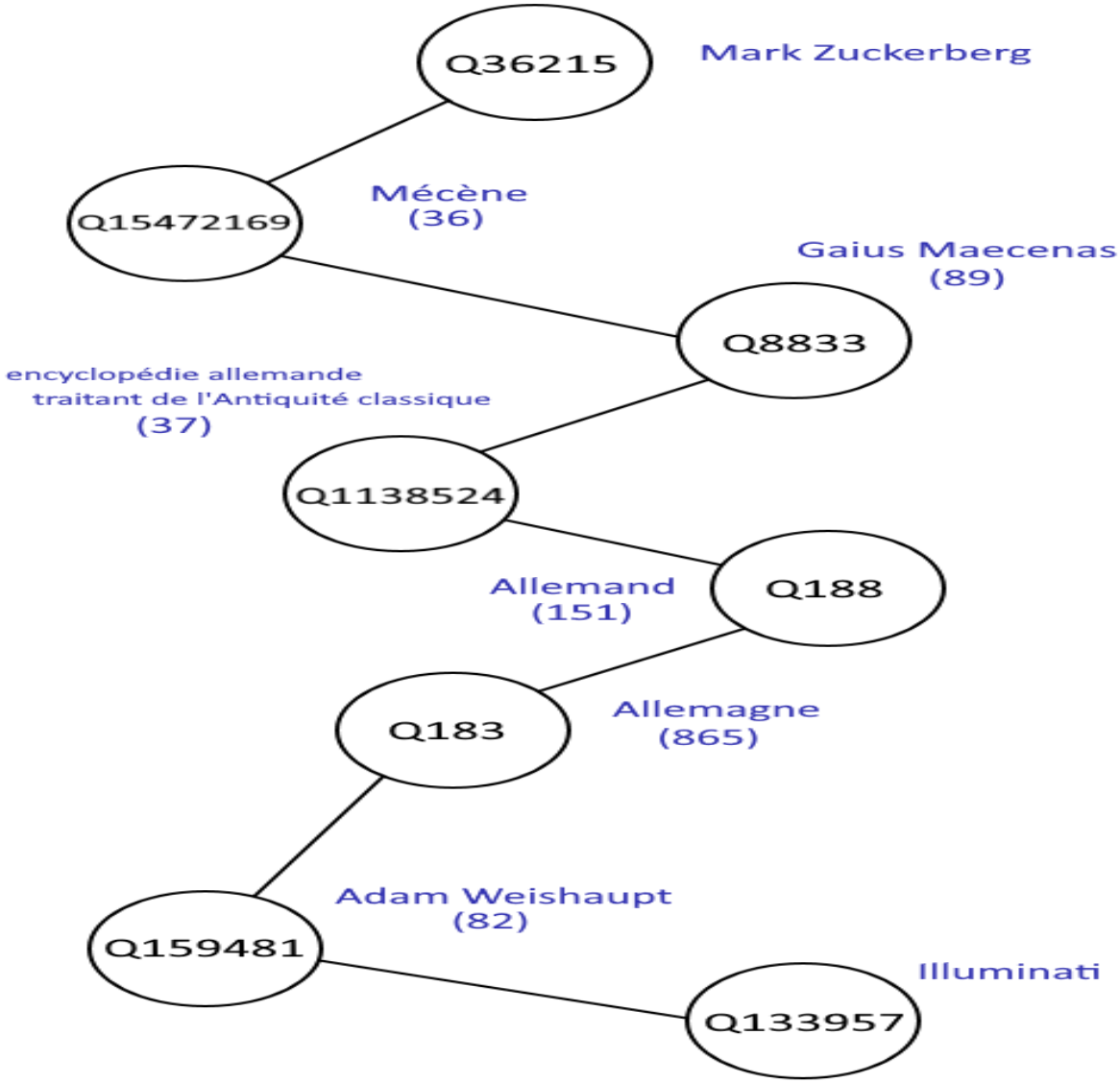
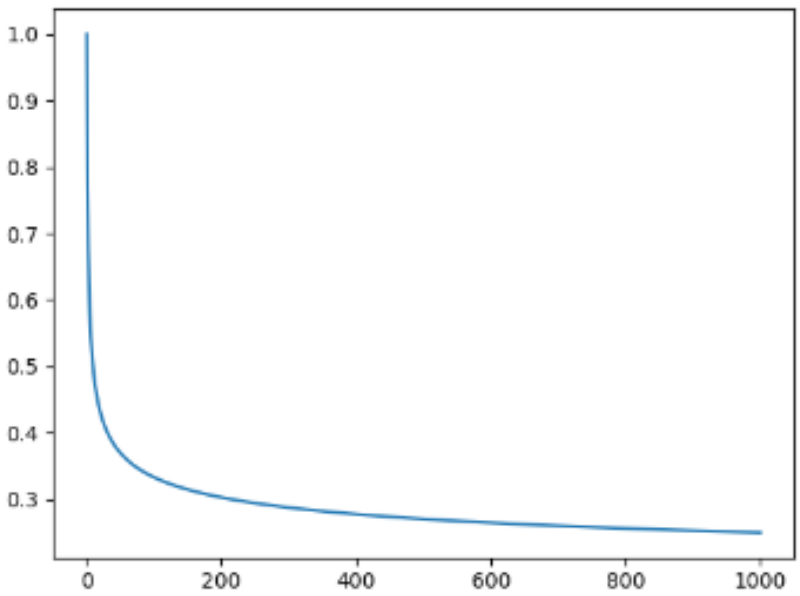


Figure 7: Un chemin entre Mark Zuckerberg et les Illuminatis

- On associe à chaque proposition p, $v(p)$ la valeur de vérité de p
- $v(p) \in [0, 1]$
- Soit $N = max_{i \in [|1,n|]}\{(N_i)\}$
  Alors
$$v(p) = \frac{1}{1 + K(N)log(N)}$$
- $K(N) = 6,8.10^{-3} \times N + 0,2$
- $v(\text{"Mark Zuckerberg est Illuminati"}) = \frac{1}{1+K(865)log(865)} = 0.05301402782046977 4$



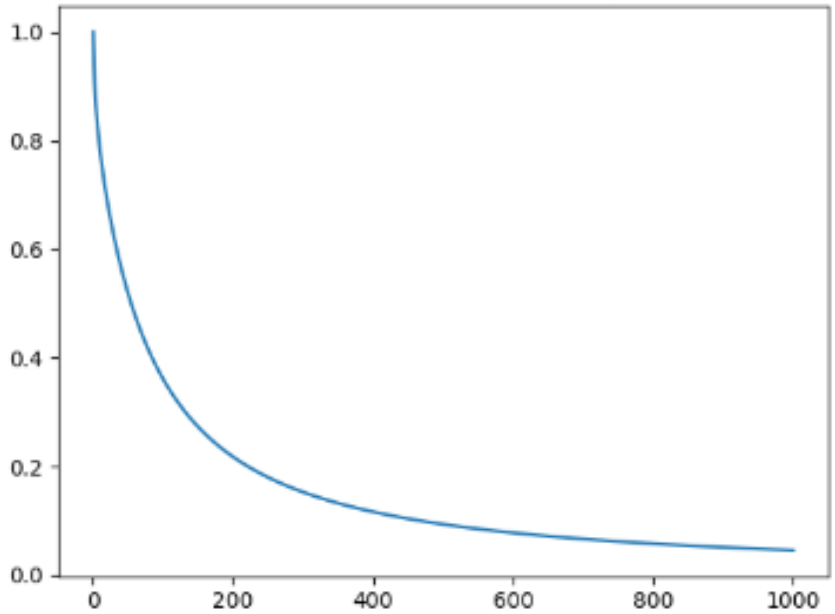


Figure 8: Comparaison sans/avec calibrage

Réflexion et conclusion

Annexe

```
from qwikidata.sparql  import return_sparql_query_results
import networkx as nx
import spacy
from json.decoder import  JSONDecodeError
from math import log10
nlp = spacy.load("fr_core_news_sm")


def filter(target):
    if target.isdigit():
        return None
    doc = nlp(target)
    sentence_ents = [str(ent) for ent in doc.ents if ent.label_]
    sentence_punct = [None for token in doc if token.is_punct]
    if len(sentence_ents) != 0 and len(sentence_punct) == 0:
        if target[:7] != "http://":
            return target
```

```
def sujet_to_wd(sujet):
    sujet_str = '"' + sujet + '"'
    query_string = """
    SELECT ?item
WHERE
{
  ?item rdfs:label """ + sujet_str + """@fr
}
    """
    res = return_sparql_query_results(query_string)
    for row in res["results"]["bindings"]:
        try:
            value = row["item"]["value"]
            if value[:31] == "http://www.wikidata.org/entity/":
                return value[31:]
        except IndexError:
            error = True
```

```
def wikidata_query(sujet_wd):
    query_string = """
    SELECT distinct ?wdLabel ?ps_Label {
      VALUES (?sujet) {(wd:""" + sujet_wd + """)}
      # Item Property Value
      ?sujet ?p ?statement .
      ?statement ?ps ?ps_ .
      ?wd wikibase:claim ?p.
      ?wd wikibase:statementProperty ?ps.
      OPTIONAL {
      ?statement ?pq ?pq_ .
      ?wdpq wikibase:qualifier ?pq .
      }
      SERVICE wikibase:label { bd:serviceParam wikibase:language "fr" }
    } ORDER BY ?wd ?statement ?ps_
    LIMIT 50
          """
```

```
res = return_sparql_query_results(query_string)
    source = []
    relations = []
    target = []
    for row in res["results"]["bindings"]:
        target_nl = row["ps_Label"]["value"]
        if filter(target_nl) != None and filter(target_nl) != "":
            try:
                node_wd = sujet_to_wd(target_nl)
            except JSONDecodeError:
                error = True
                node_wd = None

            if node_wd != None and node_wd != sujet_wd and node_wd != "
                source.append(sujet_wd)
                relations.append(row["wdLabel"]["value"])
                target.append(node_wd)

    return [sujet_wd, target, relations, len(target)]
```

```
def has_same_element(L1,L2):
    for x in L1:
        for y in L2:
            if x == y:
                return True
    return False

def build_graph(sujet_wd,target_wd):
    QUERY1 = [wikidata_query(sujet_wd)]
    QUERY2 = [wikidata_query(target_wd)]
    LIST1 = QUERY1[0][1]
    LIST2 = QUERY2[0][1]
    while not has_same_element(LIST1,LIST2):
        visited = []
        tmp1 = []
        tmp2 = []
```

```
        for x in LIST1:
            if x not in visited:
                visited.append(x)
                if x != None:
                    try:
                        query = wikidata_query(x)
                        tmp1 = tmp1 + query[1]
                        QUERY1.append(query)
                    except JSONDecodeError:
                        error = True
        for y in LIST2:
            if y not in visited:
                visited.append(y)
                if y != None:
                    try:
                        query = wikidata_query(y)
                        tmp2 = tmp2 + query[1]
                        QUERY1.append(query)
                    except JSONDecodeError:
                        error = True
        LIST1 = LIST1 + tmp1
        LIST2 = LIST2 + tmp2
    return QUERY1 + QUERY2
```

```python
def valeur_verite(sujet_wd,target_wd):
    Q = build_graph(sujet_wd,target_wd)
    T0 = []
    T1 = []
    T2 = []
    for x in Q:
        for i in range(x[3]):
            T0.append(x[0])
            T1.append(x[1][i])
            T2.append(x[2][i])
    G = nx.Graph()
    for i in range(len(T0)):
        G.add_edge(T0[i], T1[i])
    p = nx.shortest_path(G,sujet_wd, target_wd)
    N = max([wikidata_query(x)[3] for x in p[1:len(p)-1]])
    a = 6.8 * 10 ** (-3)
    b = 0.2
    return 1/(1+(a*N+b)*log10(N))
```