Projet de Web sémantique - M2 DataScale

Recherche par mots clés dans un graph RDF

Présenté par :
Walid YAICI
Roberto PETOH TSENE
Kimba SABI N'GOYE
SIYABEND Turgut

Encadrée par :
Zoubida KEDAD

8 Mars 2024

Table de Matière

1. Introduction

1.1. Description de la problématique

Etant donné un graph rdf et un ensemble de mots clés on veut extraire les ressources, propriétés et valeurs qui match le mieux avec ces mots clés et ensuite construire un graph à partir des éléments qui match

2. Solution naïve

2.1. Etape 1: recherche de ressources contenant les mots clés

- Lorsque nous recevons les mots clés.
- On parcourt les triplets du graph RDF et si l'un des mots clés de notre liste est présent en tant que Ressource, propriété ou valeur, on l'enregistre.

Voici le pseudo algorithme pour la recherche des noeuds et prédicats clés:

Algorithme 1: Recherche de mots clés dans le graph rdf

```
   Inputs: keywords (tableau de mots clés)
           graph (graph rdf sous forme de triplet sujet, prédicat, objet)
   Output: result (pour chaque mot clé une list de noeuds, d'arêtes et de littéraux)
1  result = [ ]
2  // Parcour du graph rdf complet pour chaque mot clé
   For keyword in keywords:
3      sub_graphs = [ ]
4      For subject, predicate, obj in graph:
5          If keyword.lower() in str(subject).lower():
6              sub_graphs ['nodes'].add(subject)
           End
7          If keyword.lower() in str(predicate).lower():
               sub_graphs ['predicates'].add({"predicate": predicate,
8                                             "node_src": subject,
                                              "node_dest": obj})
9          End
10         If keyword.lower() in str(obj).lower():
11             sub_graphs ['nodes'].add(obj)
12         End
13     End
14     result.add(sub_graphs)
15 End
```

2.2. Étape 2 : construction d'un nouveau graphe RDF

Afin de construire un graph à partir des éléments rdf tirés lors de la première étape nous allons suivre les étapes suivantes:

- On choisit un nœud de départ aléatoirement parmi la liste des nœuds tirés des mots clés.
- Tant que tous les noeuds clés n'ont pas été visités:
    - si le noeud en cours fait partie d'un prédicat clé alors:
        - On cherche le plus court chemin entre le nœud source et les autres nœuds clés qui n'ont pas encore été visités.
        - on fait de même avec le noeud destination du prédicat.
        - À l'issue, on choisit le plus petit court chemin des deux.
        - On ajoute l'arête à la liste des arêtes du graph résultat.
        - On ajoute les nœuds source et destination à la liste des nœuds clés visités.
    - si le noeud en cours ne fait pas partie d'un prédicat clé alors:
        - On cherche le plus court chemin entre le nœud et les autres nœuds clés qui n'ont pas encore été visités.
        - On ajoute les nœuds source et destination à la liste des nœuds clés visités.
- On ajoute toutes les arêtes du plus court chemin trouvé à la liste des arêtes résultat.

```
8   calcule le plus court chemin entre node et tous les noeud clés (excepté ceux qui ont déjà
    été considéré comme traité) (exception est faite pour le dernier noeud qui doit forcément
    être relié à un noeud clé mais qui à été traité.
9   nodes_done.add(node)
10  End
    If cas == 1:
11  shortest_path1 = shortest_path(node, sub_graphs_nodes, nodes_done, last_node )
    shortest_path2 = shortest_path(node2, sub_graphs_nodes, nodes_done, last_node )
    shortest_path = min (shortest_path1, shortest_path2)
12  nodes_done.add(node)
    nodes_done.add(node2)
13  End
14  prédicats.add(shortest_path) //rajouter toutes les arêtes du shortest_path
    node = shortest_path[len(shortest_path)] //la dernière arête du shortest path contient le
    prochain nœud clé à traiter.
15  If len(nodes_done) == len(sub_graphs_nodes) - 1: last_node = True
16  End
```

```
6   voisin des noeuds source et destination de l'arete et prendre le plus court des deux)
    node = pred['node_src']
    node2 = pred['node_dest']
    End
7   shortest_path = [ ]
    If cas == 0:
    shortest_path = shortest_path(node, sub_graphs_nodes, nodes_done, last_node ) // on
```

3. Etat de l'art sur la recherche par mot clé dans les graph RDF

3.1. Keyword Searching and Browsing in Databases using BANKS

L'article propose une approche permettant de rechercher des mots clés dans un graphe et de retourner le résultat sous forme d'arbre.

Le graph de base utilisé est obtenue à partir des tables d'une base de données relationnelle.

- Chaque tuple de la base de données relationnelle est représenté par un nœud dans le graphe.
- Les relations entre les tuples, telles que les liens entre clé étrangère et clé primaire, sont représentées sous forme d'arêtes dirigées entre les nœuds correspondants.
- Les informations telles que le nom de la table, les vues deviennent des propriétés du nœud.

3.1.1. Étape 1 : recherche de noeud pertinents

- L'approche utilisée est similaire à la nôtre.
- La différence est que le mot clé est comparé aux propriétés du nœud (noms de colonne, nom de table , vue, valeurs, …) nous comparons le mot clé aux valeurs en chaine de caractère (la ressource, la propriété, la valeur).
- Pour chaque terme de recherche dans la requête, on trouve l'ensemble des nœuds qui match.

3.1.2. Étape 2 : construction d'un arbre de réponse

Une réponse à une requête (ensemble de mots clés) est un arbre orienté enraciné contenant au moins un nœud de chaque mot clé.

On peut avoir plusieurs réponses pour une requête. Ils utilisent un ranking pour classer les réponses.

- L'idée est de trouver un sommet commun à partir duquel un chemin direct existe vers au moins un nœud pour chaque mot clé.
- Pour trouver le nœud racine, BANKS utilise un algorithme spécial qui examine la proximité de chaque nœud par rapport aux nœuds de mots clés et son importance en fonction de ses connexions avec d'autres nœuds.
- L'algorithme sélectionne ensuite le nœud ayant le score de pertinence le plus élevé comme nœud racine.

- Chaque résultat est un arbre contenant des tuples de nœuds (y compris des nœuds intermédiaires) ainsi que les nœuds resp. noms de tables et noms de colonnes
- Chaque arbre de réponses doit se voir attribuer un score de pertinence et les réponses doivent être présentées par ordre décroissant de ce score.

3.1.3. Comparons notre méthode et BANKS

Aspect	Notre méthode	BANKS
type de données	Graph RDF	données relationnelles transformées en graph
comparaison des mots clés ki	• Est-ce que la valeur en chaîne de l'URI () contient ou est égale à Ki ? • Est-ce que ces URI contiennent partiellement ou totalement un synonyme de Ki ?	• Ki = nom table | colonne | vue | relation
Construction de l'arbre de réponse	commence par un nœud aléatoire et cherche le chemin le plus court vers le chemin le plus optimal. Donc plus rapide mais chemin moins optimal	Cherche le noeud le plus proche d'au moins un noeud pour chaque mot clé Algorithme plus lent mais chemin plus court
Ranking des réponses	graph score = nombre de noeuds pertinents / total de noeuds dans le graph	graph score : combinaison du score du noeud (Ns) et score des arrêtes (Es). score = (1-a)Ns - a*Es (Ns) proportionnelle au nombre d'arrêtes Es : 1 en général Ne tient pas compte du score des noeuds intermédiaire

3.2. Keyword-Based Navigation and Search over the Linked Data Web

La problématique posée concerne l'utilisation des approches de recherche par mots-clés sur les graphes dans le contexte du Web de données liées (Linked Data web).

La problématique principale est donc de trouver une manière d'effectuer des recherches par mots-clés directement sur le Web de données liées en temps réel, plutôt que de s'appuyer sur des index locaux.

L'algorithme présenté vise à résoudre cette problématique en permettant aux utilisateurs de naviguer entre les documents en spécifiant des mots-clés qui sont ensuite utilisés pour faire correspondre des triplets RDF. L'idée est de créer une approche de recherche dynamique qui fonctionne directement sur le Web de données liées, évitant ainsi la nécessité de copier localement les graphes RDF.

3.2.1. Description de la méthode

- Les premiers mots-clés sont donnés au système par les utilisateurs ou les agents qui initient leur navigation en fournissant une liste d'emplacements de départ (seed URIs). La navigation commence par la résolution des seed URIs, après quoi les utilisateurs sont invités à introduire un mot-clé. Ce mot-clé est utilisé pour rechercher du contenu pertinent par rapport aux informations récupérées.
- Dans la section d'évaluation, ils effectuent une recherche à l'aide d'un algorithme de similarité des chaînes de caractères par rapport à la représentation de la chaîne de chaque triple
- Pour permettre d'autres correspondances, nous étendons la recherche au contenu acquis en résolvant les URI prédicats associés à la ressource actuelle. Lors de la mise en correspondance des liens RDF, nous ne sommes donc pas limités à la mise en correspondance des tokens dans la chaîne URI elle-même, mais nous pouvons également mettre en correspondance les étiquettes, les commentaires, etc.
- Une fois les triples correspondants trouvés, ils sont envoyés vers un flux de sortie, qui devient le flux d'entrée pour l'élément suivant du pipeline.
- Cela permet à la navigation de continuer à utiliser de nouveaux mots-clés, même lorsque d'autres éléments du pipeline sont encore occupés à rechercher des résultats pertinents.

3.2.2. Comparons notre méthode et keyword navigation, search

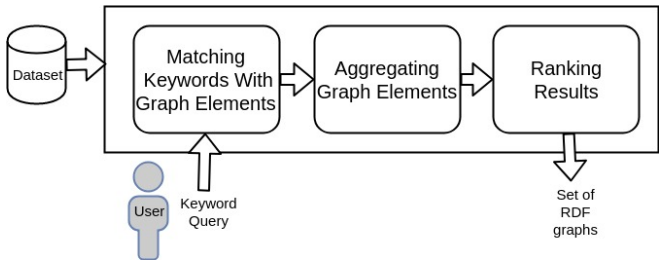
Aspects	Notre méthode	Keyword navigation, search
problématique	recherche sur un graph rdf à partir de mots clés	Naviguer entre les documents en suivant les réponses des requêtes sur un graph RDF
Type de données	base de données RDF	données web liées
comparaison entre mots clés et éléments du graph	• Est-ce que la valeur en chaîne de l'URI (predicat | propriété valeur) contient ou est égale à Ki ? • Est-ce que ces URI contiennent partiellement ou totalement un synonyme de Ki ?	garde ce module flexible afin de pouvoir utiliser différentes approches. Dans le test, compare Ki avec commentaires, labels, chaine de l'URI (prédicat | propriété | valeur)

3.3. Keyword Search Over RDF Graphs Using WordNet

Problématique:

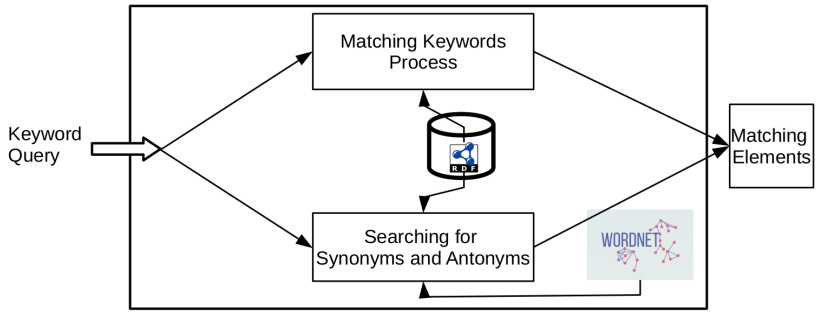
- Propose une approche de recherche par mots-clés sur les données RDF qui renvoie les meilleurs sous-graphes correspondants comme réponse à cette requête
- utilise une source externe de connaissances fournissant des relations sémantiques afin de trouver les éléments de l'ensemble de données qui correspondent aux mots-clés de la requête
- classe l'ensemble des réponses possibles à l'aide d'une méthode de classement basée sur les relations sémantiques qui ont été utilisées au cours du processus de mise en correspondance.

La solution



3.3.1. correspondance des mots clés

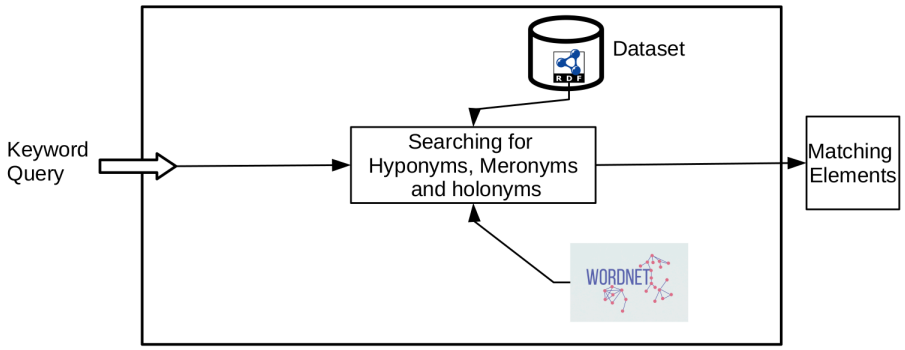
Correspondance exacte:



- Rechercher des propriétés, des ressources ou des classes qui portent le même nom que le mot-clé considéré dans l'ensemble de données.
- Interroger WordNet pour extraire les relations sémantiques (synonymes, antonymes et hypernymes) avec ki (un mot-clé) et trouver la correspondance exacte de ces relations sémantiques dans le graphique de données.

Si la recherche d'éléments correspondant exactement au mot-clé échoue, une recherche d'éléments proches est effectuée. Si un élément correspondant est trouvé dans l'ensemble de données pour chaque mot-clé, le processus de recherche s'achève.

Correspondance approchée:



- Interroger l'hyponymie, l'holonymie et la méronymie de chaque ki dans Wordnet
- Trouver la correspondance exacte de ces relations sémantiques dans le graphe de données.

Les éléments de correspondance proche sont recherchés pour chaque mot-clé sans élément de correspondance exacte. Si un mot-clé de la requête n'a pas d'éléments de correspondance

exacte ni d'éléments de correspondance proche, cela signifie qu'il n'y a pas de réponse à la requête.

3.3.2. Construction du Graph résultat

- Chaque réponse est un sous-graphe minimal connecté qui contient, pour chaque mot-clé, un élément correspondant.
- dériver les combinaisons possibles en calculant le produit cartésien des différents ensembles d'éléments correspondants
- pour chaque combinaison, le sous-graphe minimal connecté contenant les éléments correspondants de la combinaison considérée est déterminé en introduisant certains nœuds qui ne correspondent pas à des éléments correspondants et en utilisant l'algorithme de Dijkstra.
- L'algorithme d'extraction d'un sous-graphe commence par la sélection aléatoire d'un des éléments correspondants et trouve ensuite tous les chemins les plus courts entre cet élément correspondant et tous les autres éléments correspondants de la combinaison.
- Chaque sous-graphe est une réponse possible à la requête initiale.

3.3.3. Ranking des résultats

Le résultat du processus de mise en correspondance est un ensemble de sous-graphes qui sont des réponses possibles à la requête.

$$\text{Score} = 1 - \frac{[w_a * A + (1 - w_a) * L]}{N}$$

Le score ci-dessus exprime que :

- moins il y a d'éléments de liaison dans un sous-graphe, meilleure est la solution.
- Plus il y a d'éléments correspondant exactement dans un sous-graphe, meilleure est la solution.

3.3.4. Comparons notre méthode et Keyword Search Over RDF Graphs Using WordNet

Aspects	Notre approche	cette approche
Type de données	graph rdf	graph rdf
comparaison des mots clés Ki	• Est-ce que la valeur en chaîne de l'URI () contient ou est égale à Ki ? • Est-ce que ces URI contiennent partiellement ou totalement un synonyme de Ki ?	Même mode de comparaison
Recherche des synonymes	Nous donnons notre dataset à un LLM et le mot clé Ki pour lequel nous n'avons pas trouvé de matching via une API. Le LLM étant entraîné sur un grand ensemble données peut facilement nous dire si notre dataset contient un synonyme du mot. Le LLM peut aussi gérer facilement les erreurs d'orthographe	Utilise WordNet pour obtenir : synonyme, antonyme, Hyponyme, … du mot clé pour lequel il n'y a pas de matching exact
construction du graph de réponse	Choisis un nœud obtenu par matching et cherche le chemin le plus court vers le prochain nœud et ainsi de suite. La méthode est plus rapide mais le chemin global final n'est pas le plus court	Choisis l'un des nœuds obtenu par matching et cherche le chemin le plus court vers tous les autres nœuds du sous graph. Le calcul de tout chemin est plus coûteux mais on obtient un chemin global optimal
Méthode de ranking	graph score = nombre de noeuds pertinents / total de noeuds dans	introduit la notion de poids pour chaque noeud faisant parti du graph $\text{Score} = 1 - \frac{[w_a * A + (1 - w_a) * L]}{N}$ wa ici est le poid de A

	le graph	A : nombre de noeud matching L : nombre de noeud intermédiaire

3.4. Survey

Basée sur des méthodes de stockage :

interrogation par mot-clé de données RDF centralisées

- interrogation par mot-clé d'une base de données relationnelle
    - opérations d'auto-jointure
- requête par mot-clé basée sur un graphe
    - réduire l'espace de recherche grâce à des index efficaces
    - limiter l'étendue de la recherche à des sous-graphes plus petits
    - trouver les résultats de recherche correspondants dans les sous-graphes

Interrogation par mot-clé sur des données RDF distribuées

Requête par mot-clé par traduction de requête

- Extraire des informations sur les motifs RDF des données RDF
- Mettre en place un index sommaire pour la traduction des requêtes
- L'index sommaire est utilisé pour trouver les relations entre les entités contenant les mots-clés.
- Le sous-graphe contenant les relations est converti en un langage d'interrogation formel.
- Trier et renvoyer le résultat à l'utilisateur
- L'utilisateur sélectionne une requête satisfaisante à exécuter par le moteur de recherche existant et les résultats finaux sont obtenus.

4. Dataset de test

Afin de tester notre approche de recherche par mots clés nous avont éxécuter la requête suivante sur DbPedia afin d’avoir 4 films avec le propriétés title, release_date, Director,

Starring et type et pour les acteurs qui ont participés (starring) les propriétés name, birth_date et type.
Voici la requête Sparql qu'on a exécuté pour avoir ces informations:

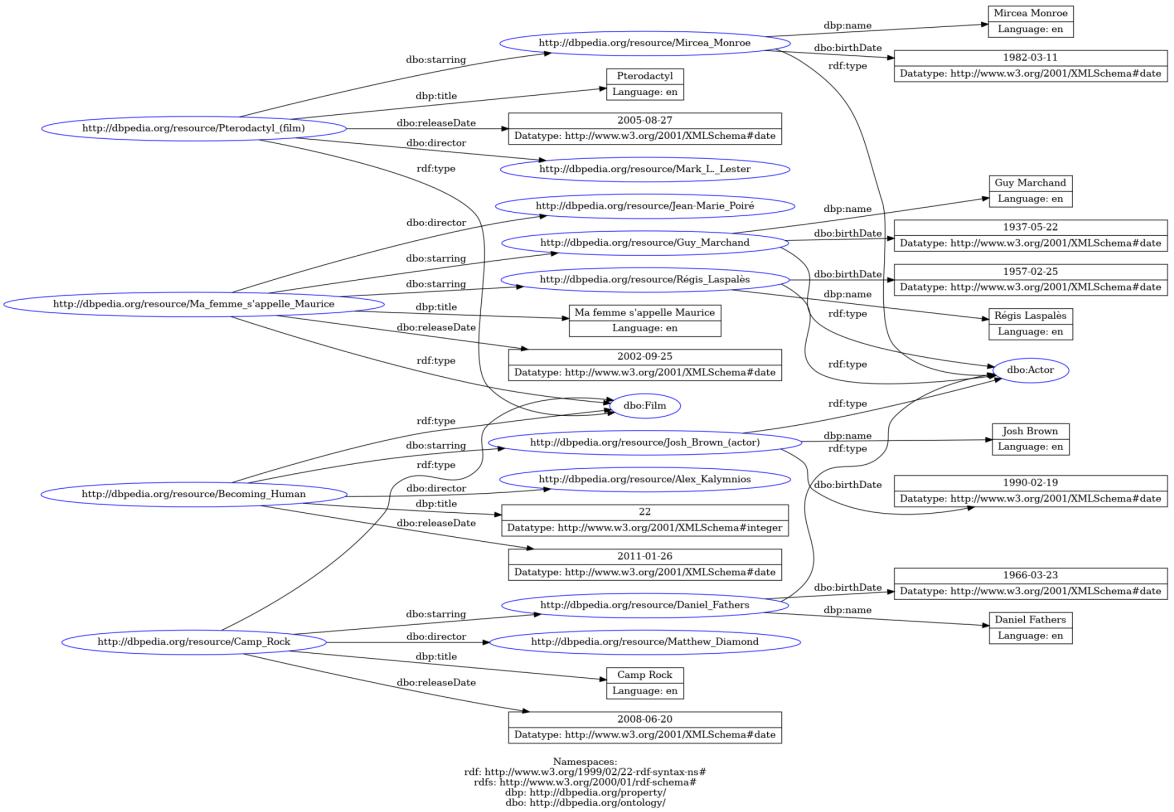
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

CONSTRUCT {
  ?film rdf:type dbo:Film ;
        dbp:title ?title ;
        dbo:director ?director ;
        dbo:starring ?actor ;
        dbo:releaseDate ?releaseDate .

  ?actor rdf:type dbo:Actor ;
         dbp:name ?actorName ;
         dbo:birthDate ?birthDate .
}
WHERE {
  ?film rdf:type dbo:Film ;
        dbp:title ?title ;
        dbo:director ?director ;
        dbo:starring ?actor ;
        dbo:releaseDate ?releaseDate .

  ?actor rdf:type dbo:Actor ;
         dbp:name ?actorName ;
         dbo:birthDate ?birthDate .
}
LIMIT 5
```

Voici le graph obtenu:



5. Proposition d'amélioration et de variante de la méthode naïve

5.1. Recherche de synonymes pour les mots clés

Pour les mots clés qui n'ont pas de correspondance dans le graph (aucun prédicat ni de nœud retourné durant la phase 1), on propose de chercher des synonymes à ces mots qui pourrait donner des résultats lors de l'étape 1 de recherche.

Afin d'avoir une méthode qui soit pertinente et qui prenne en considération le contexte du graph rdf, on propose d'utiliser une méthode basée sur les LLM.

Pour cela on utilise l'api d'OPENAI et on utilise trois type de rôles pour construire notre prompt:

- Un rôle de type assistant auquel on donne le fichier rdf sur lequel on fait notre recherche ou bien une ontologie qui résume le graph.

  (ceci va permettre au LLM de prendre le fichier rdf ou ontologie comme base de connaissances pour répondre aux questions qu'on lui posera après)

- Un rôle de type système qui permet de spécifier dans quel format on veut avoir notre réponse.
- Un user à qui on passe un prompte qui contient le mot clé pour lequel on veut trouver un synonyme accompagné d'instructions pour lui demander de trouver le synonyme idéal pour répondre à des requêtes de type recherche par mot clés dans un graph.

Voici le pseudo algorithme pour exécuter cette fonction:

Algorithme 3 : Recherche de synonyme

Inputs: *keyword (mot clé)*
graph (fichier rdf ou ontologie)
client (client OpenAi pour utiliser l'api)

Output: *result (synonyme du mot clé)*

```
1  assistant = "you use only the following context to answer.\nHere's an rdf graph in xml
   format: "+ str(graph)
   prompt_template = """give me a keyword synonyme of {} so that when i search for node,
   relation, literal coresponding to that keyword in the rdf graph given it return me some
2  results.
   return me only the keyword synonyme not rdf elements.
   the keyword must correspond to an element in rdf graph.""".format(keyword)
   messages = [{'role': 'system', 'content': 'print only the synonyme word.'},
3              {"role": "user", "content": prompt_template},
               {"role": "assistant", "content": assistant},
               ]
   result = client.chat.completions.create(
       model="gpt-3.5-turbo",
4      messages=messages,
       temperature=0
   )
```

Cette méthode permet d'avoir des synonymes qui soit vraiment en relation avec le contexte du graph rdf qu'on manipule contrairement à d'autres méthode qui cherche des synonymes sans avoir de contexte.

5.2. Produit cartésien des éléments extraits

Dans la solution naïve il arrive que pour certains mots clés on ait beaucoup de nœuds/prédicats qui match avec ce mot clé. Par exemple dans notre dataset si on veut chercher les films dans lesquels joue Régis Laspalès on pourrait utiliser les mots clés suivants: (Régis Laspalès, starring), mais en exécutant cette requête, pour le mot clé starring on aura toutes les arêtes de type starring.

Afin d'apporter une réponse qui ne soit pas saturée de relations/nœuds qu'on cherche pas vraiment, on propose d'utiliser la méthode du produit cartésien.

Cette méthode va permettre de répondre aux requêtes de recherche ou on sait d'avance que pour chaque mot clé on veut avoir seulement 1 nœud/prédicats.

- Pour chaque mot clé on va parcourir tous les noeuds/prédicats qui ont été extrait à l'étape 1
- On fait la combinaison de chaque nœud/prédicat avec tous les autres nœuds/prédicats des autres mots clés.
- Si pour chaque mot clé on a N_j noeuds/prédicats alors le nombre de combinaisons possible sera égale à $\prod N_j$

Contrairement à la méthode de base qui aura tendance à retourner plus d'information que nécessaire, cette méthode permet d'avoir des graph moins dense mais avec moins de nœuds non clés.

5.3. Méthode de ranking des graph résultats

Avec l'ajout de la méthode du produit cartésien, il est nécessaire d'introduire une méthode de ranking des graphs résultats afin de savoir quelle est la meilleure combinaison du produit cartésien.

Pour le ranking nous avons opté pour le score suivant:

$$Score = \frac{A}{N}$$

Ou A va représenter le nombre de nœuds/prédicats extrait après l'étape 1.

N va représenter le nombre de nœuds/prédicats du graph de l'exécution de l'étape 2 et donc après avoir potentiellement introduit des nœuds et prédicats non clés.

Plus le score sera élevé, meilleur est le résultat.

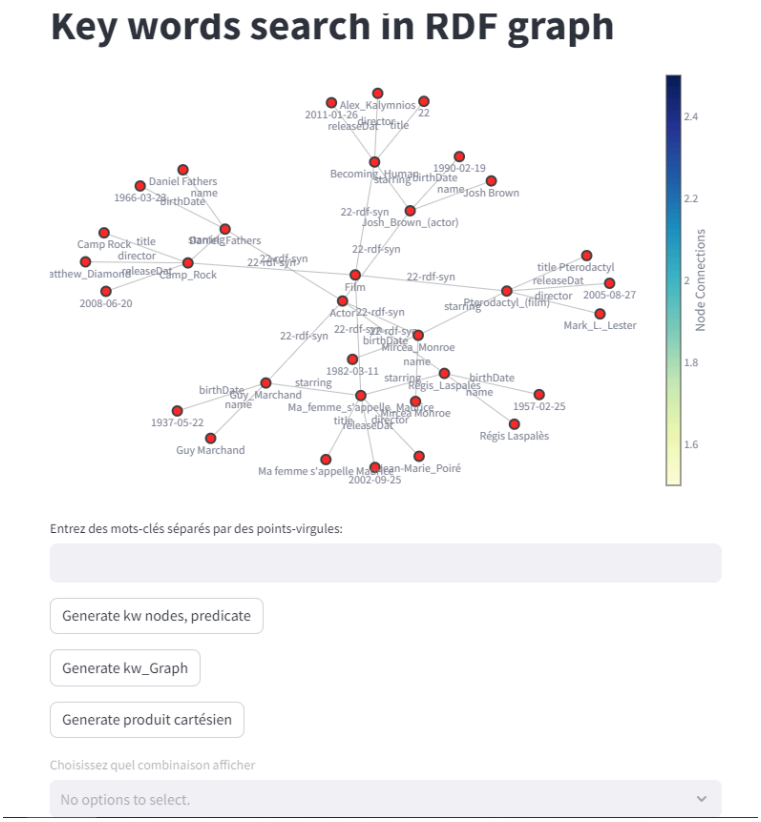
Cette méthode de ranking favorise les graph ou on introduit le moins de nœuds/prédicats non clés. Elle considère aussi que dans le cas ou un mot n'a pas eu de correspondance mais que son synonyme a pu avoir des match, alors les nœuds/prédicats synonymes sont considérés comme des nœuds clés, contrairement à d'autres méthodes qui pénalisent les nœuds synonymes.

Nous avons décidé de ne pas pénaliser les nœuds synonymes car on considère que notre méthode de recherche de synonyme basée sur le contexte du graph est pertinente et que les éléments retournés peuvent être considérés comme clés.

6. Test de la méthode naïve et des améliorations proposés avec l'interface web

6.1. Présentation de l'interface

Afin de pouvoir interroger au mieux notre graph rdf nous avons conçu une interface web avec streamlit et plotly pour l'affichage des graph.



L'utilisateur renseigne les mots clés séparés par un ';', il y a:

- un bouton pour afficher les éléments clés qui ont été extraits (après l'étape 1).
- Un bouton pour afficher le graph final de l'étape 2 avec la méthode naïve.
- Un bouton pour générer les combinaisons linéaire et qui va remplir la liste déroulante qui permet de choisir une combinaison à afficher.

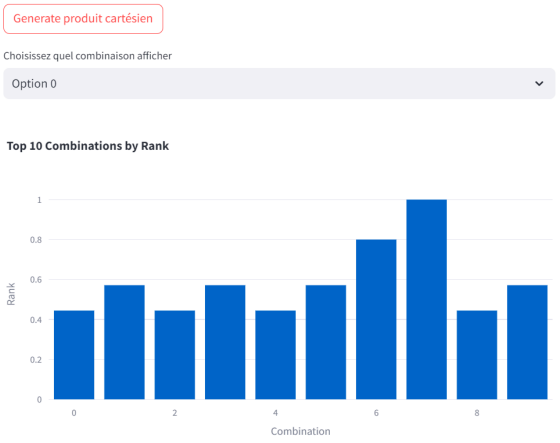
6.2. Test de requêtes

Testons quelques requêtes afin de voir les résultats obtenus:

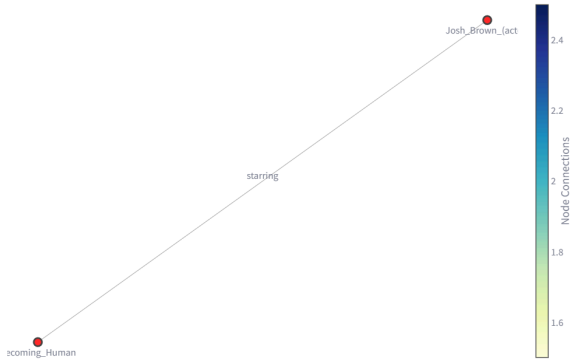
- ☐ Si dans le graph des films on veut connaître les films dans lesquels un "josh" est acteur, on peut utiliser les mots clés suivants: [Josh Starring].

voici le graph des mots clés obtenu (après etape 1):

Voici le score de ranking obtenu par les 10 meilleurs combinaisons possibles (meilleur score = 1):

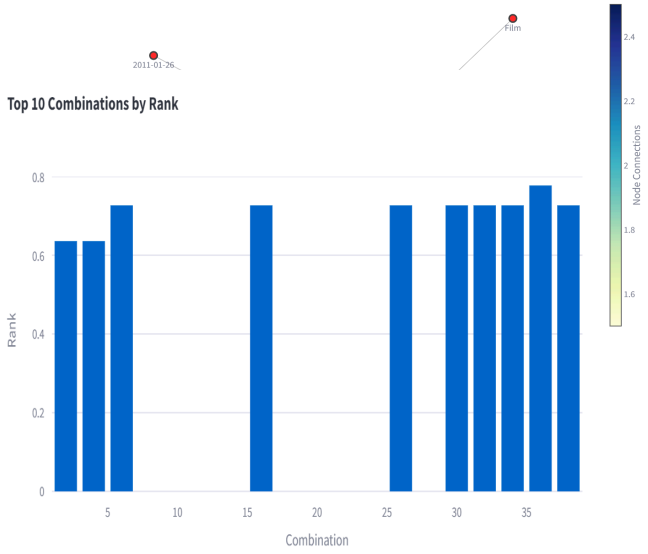


si on affiche le graph du meilleur score de rank:



Le graph obtenu représente bien l'acteur josh avec le film dans lequel il a joué.

- ☐ Si on veut connaître quel est l'acteur qui a joué dans un film sortie en 2011 et quel était le directeur du film on peut utiliser la requête suivante: [2011, Film, Starring, Director]:



Voici le ranking des 10 meilleurs solutions (meilleur score = 0,77)et le graph avec le plus haut score:

Le résultat obtenu répond bien à la requête demandée et nous retourne le film avec sa date de sortie exacte, son acteur et son directeur.

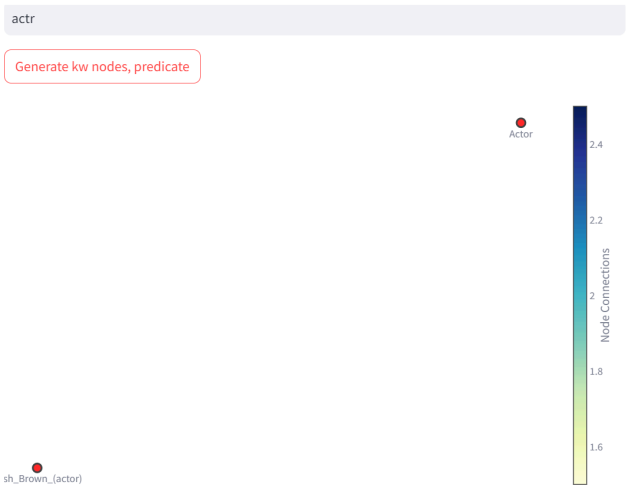
6.3. Test de la fonction de recherche de synonymes

Afin d'avoir un premier aperçu sur les améliorations qu'apporte l'utilisation de notre méthode de recherche de synonyme, on tester de chercher un mot clé en commettant des fautes d'orthographes et des mots clés synonymes de ressources présentes dans le graph:

- ☐ si on cherche des mots clés synonymes de starring tel que : 'acted in', 'played', 'acted', 'perform' ca donne le graph suivant de mots clés suivant:

ceci indique que tous ces mots il leurs a trouvé comme synonyme le mot 'starring'

- ☐ Cette approche sert aussi quand l'utilisateur rentre le mot clé avec des fautes d'orthographe, par exemple si on cherche le mot clé 'actr' il va retrouver comme synonyme 'Actor':



- On peut aussi au lieu de donner un mot clé, indiquer une description de la ressource qu'on cherche, si par exemple on ne connaît pas comment s'appel la relation qui permet d'avoir la date de naissance d'un acteur, on peut mettre comme mot clé 'date of birth' et la fonction de synonyme arrive à trouver qu'on veut parler de la relation birthDate:
- Pareil si on ne connaît pas la relation 'releaseDate', on peut donner la description

'date of film' et la fonction de synonymie nous retourne le mot clé 'releaseDate':

7. Expérimentations

7.1. Expérimentation sur le temps d'exécution

On a testé d'autres requêtes, 8 requêtes au total:

1- (["josh","starring"])

2- (["Daniel","starring"])

3- (["2005","film","starring","director"])

4- (["name","birthDate","Camp Rock"])

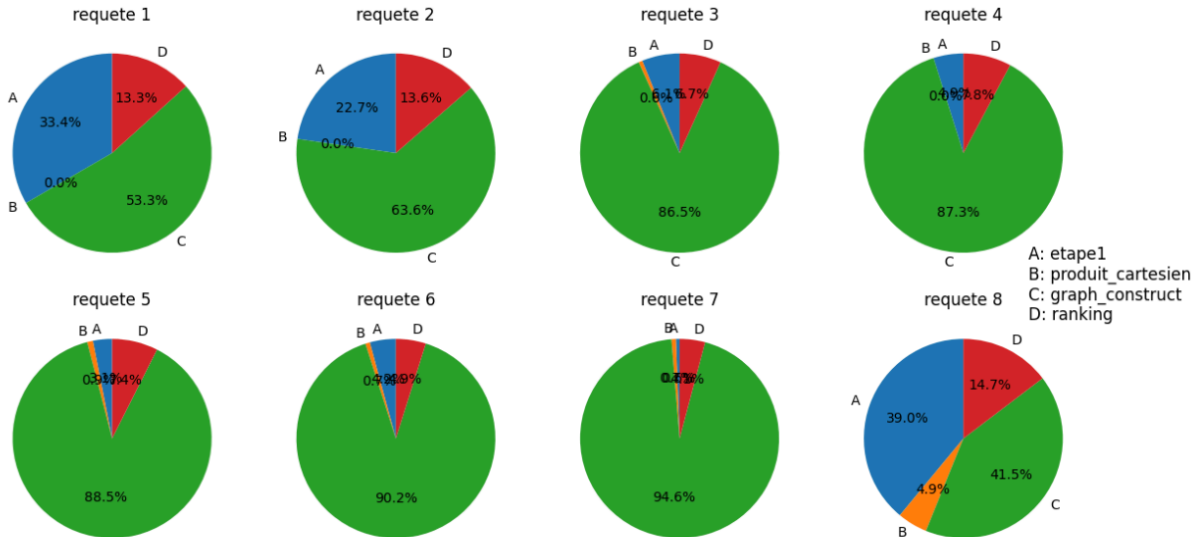
5- (["releaseDate","starring","director","Becoming_Human"])

6- (["title","starring", "Guy"])

7- (["film","starring","2002","name","birthDate"])

8- (["director","film","2002"])

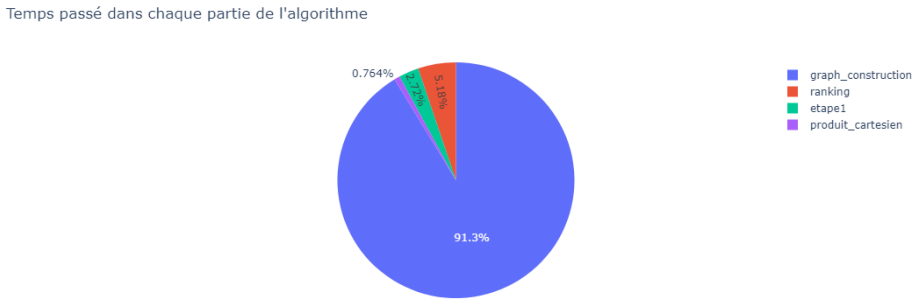
Nous représentons dans le graphique ci-dessous la part de chaque étape dans la durée totale pour chaque requête. Les lettres sur les images et les 4 étapes correspondantes sont comme suivant : 'A' : etape1, 'B' : produit_cartesien, 'C' : graph_construct, 'D' : ranking



Dans certains résultats, nous observons que la première étape prend beaucoup de temps, ce qui est dû au nombre élevé de nœuds correspondant au mot clé qui sera renvoyé par la requête. Pour certaines requêtes, la phase de construction du graphe prend beaucoup de temps, tandis que pour d'autres, cette durée est due au grand nombre de nœuds intermédiaires nécessaires pour créer les connexions entre eux en fonction du nombre de nœuds renvoyés par la première étape.

On observe que nos requêtes se répartissent en 2 groupes en fonction de la proportion de la durée de l'étape graph_construct par rapport à la durée totale. Les requêtes du premier groupe (3, 4, 5, 6 et 7) contiennent généralement des prédicats dans les mots clés, ce qui retourne un grand nombre de nœuds en résultat de la requête. La phase de construction du graphe, appelée graph_construct, prend considérablement plus de temps que les autres étapes. En revanche, pour le deuxième groupe de requêtes (1, 2 et 8), les prédicats sont généralement absents, et peu de mots clés sont impliqués, ce qui entraîne un nombre limité de nœuds en résultat de la requête. Ainsi, dans ce groupe, l'étape graph_construct occupe une part plus faible de la durée totale de la requête par rapport au premier groupe.

On a exécuté chacune des 8 requêtes 10 fois. Le résultat qu'on a obtenu est ci-dessous:

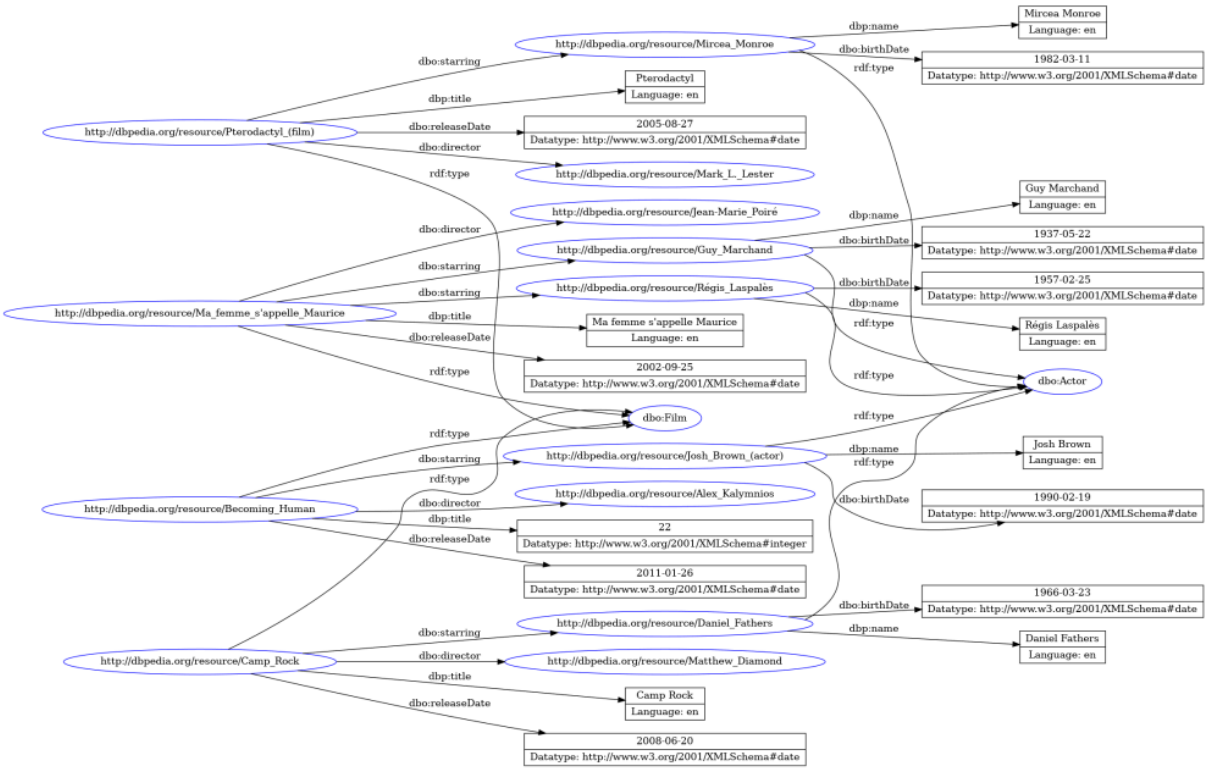


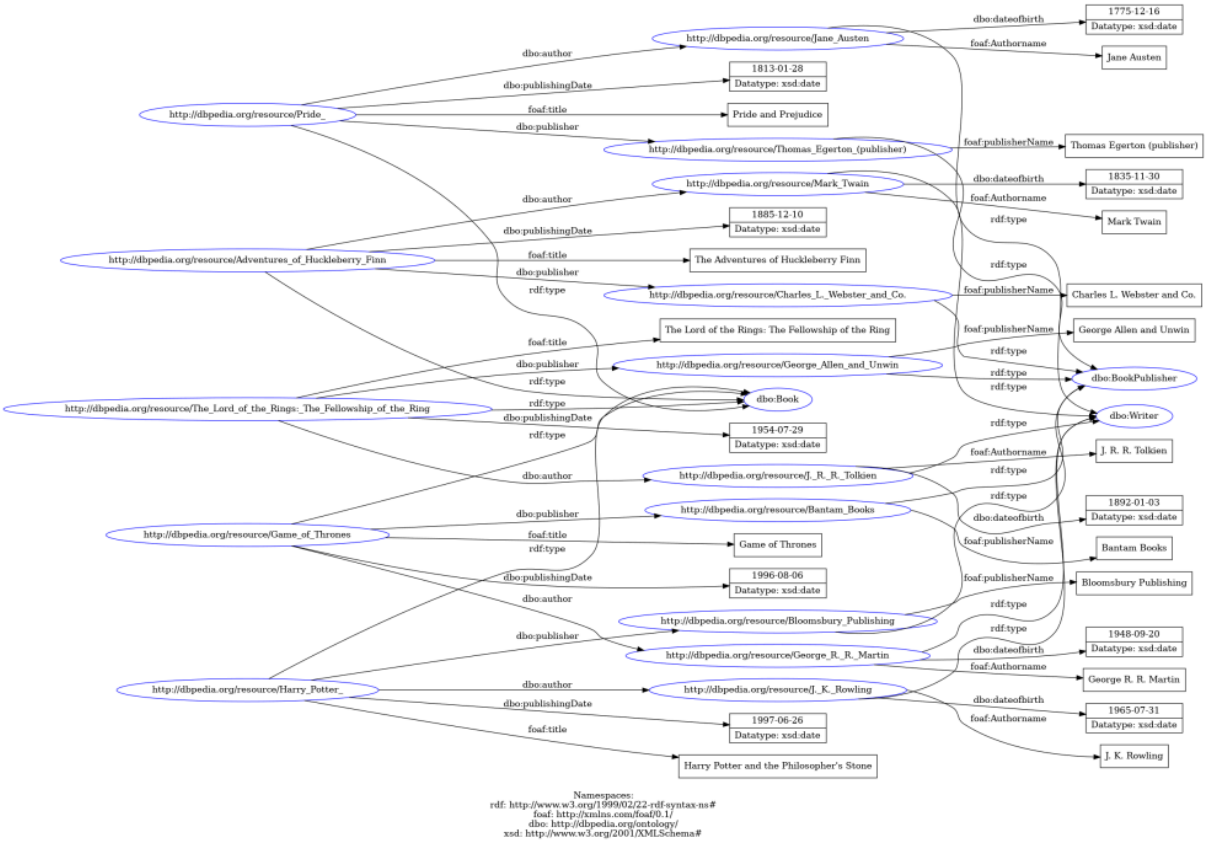
En moyenne, nous observons que l'étape graph_construct représente environ 91 % du temps total. Étant donné que la plupart de nos requêtes sont riches en prédicats, il est naturel d'obtenir un tel résultat.

7.2. Expérimentation sur la performance

Dans cette section, nous présentons les détails de nos expérimentations pour évaluer les performances de notre système de recherche basé sur les mots clés dans un graphe RDF.

Dataset
Nous avons utilisé un ensemble de données RDF contenant des informations sur des films ainsi que sur des livres, avec des ontologies associées pour chaque domaine.


http://dbpedia.org/resource/Pterodactyl_(film)
dbo:starring
http://dbpedia.org/resource/Mircea_Monroe
dbp:name
Mircea Monroe
Language: en
dbo:birthDate
1982-03-11
Datatype: http://www.w3.org/2001/XMLSchema#date
rdf:type
dbp:title
Pterodactyl
Language: en
dbo:releaseDate
2005-08-27
Datatype: http://www.w3.org/2001/XMLSchema#date
dbo:director
http://dbpedia.org/resource/Mark_L._Lester
rdf:type
dbo:Film
http://dbpedia.org/resource/Ma_femme_s'appelle_Maurice
dbo:director
http://dbpedia.org/resource/Jean-Marie_Poiré
dbo:starring
http://dbpedia.org/resource/Guy_Marchand
dbp:name
Guy Marchand
Language: en
dbo:birthDate
1937-05-22
Datatype: http://www.w3.org/2001/XMLSchema#date
http://dbpedia.org/resource/Régis_Laspalès
dbo:birthDate
1957-02-25
Datatype: http://www.w3.org/2001/XMLSchema#date
dbp:name
Régis Laspalès
Language: en
dbp:title
Ma femme s'appelle Maurice
Language: en
dbo:releaseDate
2002-09-25
Datatype: http://www.w3.org/2001/XMLSchema#date
dbo:Actor
http://dbpedia.org/resource/Becoming_Human
http://dbpedia.org/resource/Josh_Brown_(actor)
Josh Brown
Language: en
http://dbpedia.org/resource/Alex_Kalymnios
1990-02-19
Datatype: http://www.w3.org/2001/XMLSchema#date
22
Datatype: http://www.w3.org/2001/XMLSchema#integer
2011-01-26
Datatype: http://www.w3.org/2001/XMLSchema#date
http://dbpedia.org/resource/Daniel_Fathers
1966-03-23
Datatype: http://www.w3.org/2001/XMLSchema#date
Daniel Fathers
Language: en
http://dbpedia.org/resource/Camp_Rock
http://dbpedia.org/resource/Matthew_Diamond
Camp Rock
Language: en
2008-06-20
Datatype: http://www.w3.org/2001/XMLSchema#date
Namespaces:
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs: http://www.w3.org/2000/01/rdf-schema#
dbp: http://dbpedia.org/property/
dbo: http://dbpedia.org/ontology/


http://dbpedia.org/resource/Pride_
dbo:author
http://dbpedia.org/resource/Jane_Austen
dbo:dateofbirth
1775-12-16
Datatype: xsd:date
foaf:Authorname
Jane Austen
dbo:publishingDate
1813-01-28
Datatype: xsd:date
foaf:title
Pride and Prejudice
dbo:publisher
http://dbpedia.org/resource/Thomas_Egerton_(publisher)
foaf:publisherName
Thomas Egerton (publisher)
http://dbpedia.org/resource/Mark_Twain
1835-11-30
Datatype: xsd:date
Mark Twain
1885-12-10
Datatype: xsd:date
http://dbpedia.org/resource/Adventures_of_Huckleberry_Finn
The Adventures of Huckleberry Finn
http://dbpedia.org/resource/Charles_L._Webster_and_Co.
Charles L. Webster and Co.
The Lord of the Rings: The Fellowship of the Ring
George Allen and Unwin
http://dbpedia.org/resource/George_Allen_and_Unwin
dbo:BookPublisher
http://dbpedia.org/resource/The_Lord_of_the_Rings:_The_Fellowship_of_the_Ring
dbo:Book
dbo:Writer
1954-07-29
Datatype: xsd:date
http://dbpedia.org/resource/J._R._R._Tolkien
J. R. R. Tolkien
http://dbpedia.org/resource/Bantam_Books
1892-01-03
Datatype: xsd:date
http://dbpedia.org/resource/Game_of_Thrones
Game of Thrones
Bantam Books
1996-08-06
Datatype: xsd:date
Bloomsbury Publishing
http://dbpedia.org/resource/Bloomsbury_Publishing
http://dbpedia.org/resource/George_R._R._Martin
1948-09-20
Datatype: xsd:date
George R. R. Martin
http://dbpedia.org/resource/Harry_Potter_
http://dbpedia.org/resource/J._K._Rowling
1965-07-31
Datatype: xsd:date
1997-06-26
Datatype: xsd:date
J. K. Rowling
Harry Potter and the Philosopher's Stone
Namespaces:
rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
foaf: http://xmlns.com/foaf/0.1/
dbo: http://dbpedia.org/ontology/
xsd: http://www.w3.org/2001/XMLSchema#

Méthodologie

Pour évaluer notre système, nous avons défini 10 requêtes exactes pour chaque dataset et leurs équivalents faisant intervenir la fonction de recherche de synonymes. Les réponses renvoyées par notre système sont comparées aux réponses attendues que nous avons calculées manuellement. Les métriques utilisées sont: la précision, le recall, le f1-score et l'édit distance. Nous calculons ensuite la moyenne, l'écart-type, le minimum et le maximum des scores obtenus sur les 10 requêtes pour chacune des métriques.

7.3. Résultats

7.3.1. Films Dataset

Evaluation des requêtes exactes

Statistiques:

```
            Moyenne    Ecart-type   Min        Max
ED          : [1.6       1.49666295 0.         4.        ]
Precision   : [0.74354978 0.2177335  0.33333333 1.        ]
Recall      : [0.87111111 0.22084015 0.33333333 1.        ]
F1 Score    : [0.79242979 0.21101787 0.38461538 1.        ]
```

Evaluation des requêtes utilisant la fonction synonyme

Statistiques:

```
            Moyenne    Ecart-type   Min        Max
Edit distance: [1.6      1.49666295 0.         4.        ]
Precision   : [0.74354978 0.2177335  0.33333333 1.        ]
Recall      : [0.87111111 0.22084015 0.33333333 1.        ]
F1 Score    : [0.79242979 0.21101787 0.38461538 1.        ]
```

7.3.2. Livre Dataset

Evaluation des requêtes exactes

Statistiques:

```
            Moyenne    Ecart-type   Min        Max
Edit distance: [1.4      0.91651514 0.         2.        ]
Precision   : [0.76262404 0.14947639 0.55555556 1.        ]
Recall      : [0.93142857 0.13950349 0.6        1.        ]
F1 Score    : [0.83333333 0.13170885 0.6        1.        ]
```

Evaluation des requêtes utilisant la fonction synonyme

Statistiques:

	Moyenne	Ecart-type	Min	Max
Edit distance:	[1.2	0.9797959	0.	2.]
Precision :	[0.76770341	0.1629941	0.42857143	1.]
Recall :	[0.91604396	0.13957619	0.6	1.]
F1 Score :	[0.82937729	0.13989567	0.5	1.]

7.4. Discussion

7.4.1. Requêtes exactes

Les résultats pour les requêtes exactes montrent que le système est capable de fournir des réponses précises et pertinentes dans la plupart des cas. Les scores de précision et de rappel sont relativement élevés, ce qui indique que le système retourne des résultats pertinents et couvre un grand nombre de réponses attendues.

Pour le dataset Films, la moyenne du F1 Score est d'environ 0,79, ce qui montre une bonne performance globale du système. Les écarts-types modérés indiquent que la performance varie légèrement d'une requête à l'autre.
Pour le dataset Livres, la moyenne du F1 Score est d'environ 0,83, ce qui est légèrement supérieur aux résultats du dataset Films. Cela montre que le système est légèrement plus performant sur les données de livres que sur les données de films.

7.4.2. Requêtes utilisant la fonction synonyme

Les résultats pour les requêtes utilisant la fonction synonyme sont similaires à ceux des requêtes exactes, ce qui indique que la fonction de recherche de synonymes fonctionne bien et n'affecte pas significativement la performance du système.

Pour le dataset Films, la moyenne du F1 Score est d'environ 0,79, identique aux requêtes exactes. Cela montre que l'utilisation de la fonction synonyme n'a pas d'impact négatif sur la performance du système.
Pour le dataset Livres, la moyenne du F1 Score est d'environ 0,83, également similaire aux requêtes exactes. Cela montre que la fonction de recherche de synonymes est efficace et maintient la performance du système.

7.4.3. Comparaison des résultats requêtes exactes avec requêtes synonymes

La comparaison des résultats des requêtes exactes et des requêtes utilisant la fonction synonyme montre que l'utilisation de la fonction synonyme n'a pas d'impact significatif sur la performance du système. Les scores de précision, de rappel et de F1 Score sont similaires pour les deux types de requêtes, ce qui indique que la fonction de recherche de synonymes est efficace et permet d'élargir la portée des requêtes sans compromettre la qualité des résultats.

8. Conclusion

Le système montre de bonnes performances sur les deux datasets (Films et Livres) pour les requêtes exactes et les requêtes utilisant la fonction synonyme. Les scores de précision, de rappel et de F1 Score sont généralement élevés, ce qui indique que le système est capable de fournir des réponses précises et pertinentes. L'utilisation de la fonction synonyme n'affecte pas significativement la performance du système, ce qui montre que cette fonction est efficace pour élargir la portée des requêtes sans compromettre la qualité des résultats. La distance d'édition, qui mesure la similitude de la structure des deux graphes, est également similaire entre les requêtes exactes et les requêtes synonymes, ce qui indique que la fonction synonyme maintient la structure des graphes de résultats.