Rapport de Stage : Laboratoire I3S/CNRS

Représentation de l'état des connaissances de l'utilisateur dans le cadre de la recherche en tant qu'apprentissage : prise en compte des connaissances implicites

Arno LESAGE SD2A

Établissement d'envoi : IUT Côte d'Azur (département SD) – Université Côte d'Azur
Établissement d'accueil : Laboratoire I3S/CNRS – Université Côte d'Azur

Maîtres de stage : Mme. Célia DA COSTA PEREIRA, M. Andrea TETTAMANZI
Tuteur de stage : Mme. Célia DA COSTA PEREIRA

Période de stage : À partir du 2 avril 2024, jusqu'au 2 juillet 2024

Sommaire

I – Présentation du stage

A – Introduction et contexte

Au cours de cette deuxième année de BUT SD, j'ai eu la chance de pouvoir effectuer mon stage au sein du laboratoire I3S/CNRS **[00]**. Deuxième stage que j'effectue dans cet organisme, le premier était un stage de découverte du monde de la recherche à partir d'un projet consistant à essayer d'adapter ELIZA **[01]** (le premier Chatbot « moderne », publié entre 1964 et 1966 par Joseph Weizenbaum) se basant initialement sur un système de patrons/mots clés, vers un système plus flexible faisant usage de Word Embedding pré entrainé (plongements lexicaux) **[02]**.

Ce second stage est donc décisif pour moi, car il me permettra, dans la continuité du premier, de confirmer ou non mon attirance pour la recherche. Celui-ci ne se consacre plus sur l'adaptation d'un Chatbot, mais sur un domaine s'appelant « Search As Learning » (SAL), soit en français la recherche en tant qu'apprentissage.

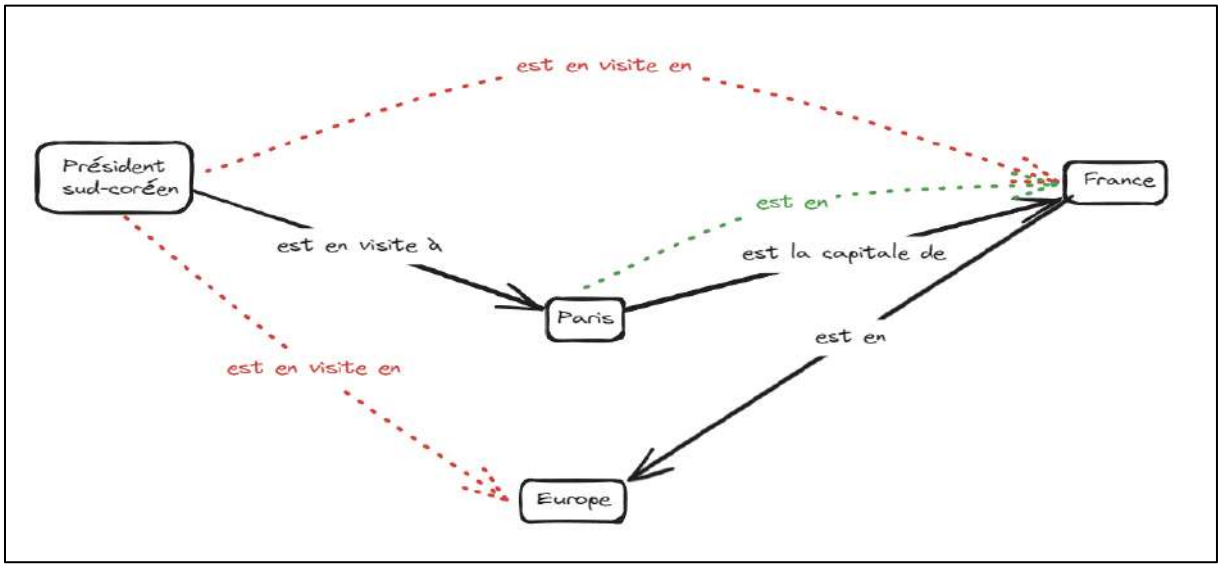
Ce domaine d'études cherche à « conceptualiser la recherche d'information ». Cette recherche d'information se fait généralement de façon exploratoire et a pour but pour l'utilisateur d'acquérir des connaissances. L'une des problématiques du domaine est donc d'essayer de « mesurer » ce gain de connaissance lors d'une session de recherche de l'utilisateur pour savoir à quel point celui-ci se rapproche de ses objectifs d'apprentissage.

Le stage confié se base ainsi sur des travaux précédemment publiés par le laboratoire en 2023 par Mme. Dima El Zein (ex-I3S), Mme. Célia da Costa Pereira (I3S), M. Andrea Tettamanzi (I3S, INRIA), Mme. Cathy Escazut (I3S) et Arthur Câmara (Université de technologie de Delft), parmi lesquelles : la création d'un cadre pour la représentation des connaissances de l'utilisateur en se basant sur la reconnaissance d'entités dans le texte ($RULK_{KW}$) **[03]** et des entités nommées ($RULK_{NE}$) **[06]**. Mais aussi sur des travaux de M. Hadi Nasser (I3S) en 2024 et sa nouvelle approche considérant à la fois les entités et leurs relations afin d'estimer le gain de connaissance de l'utilisateur ($RULK_{KG}$) **[04]**. Lors de ces études, il a été conclu que les deux approches apportent des informations complémentaires sur la représentation des connaissances de l'utilisateur.

Malgré tout, cette nouvelle approche ne prend pas en compte le fait que l'utilisateur peut avoir des connaissances initiales sur certaines relations, cela inclut notamment la transitivité des relations et la subsomption de ces dernières.

B – Explications et définitions

Afin de définir et d'expliquer le concept de transitivité et de subsomption de manière simple, voici un exemple de ce que l'on pourrait obtenir à partir d'un texte avec $RULK_{KG}$.



Ici, les entités et les arcs qui les relient ne sont présents dans celui-ci qu'uniquement parce qu'il y a eu un lien explicite tel que « la France est en Europe » dans le texte qui a permis de le générer.

Ce graphe ne permet donc pas de comprendre directement que Paris est en France, mais seulement que Paris est la capitale de la France, or l'utilisateur peut savoir implicitement que la capitale d'un pays est dans ce pays, nous pouvons donc remplacer l'arc « est la capitale de » par l'arc « est en » dans le triplet <Paris, est la capitale de, France>, il s'agit donc d'une subsomption.

De la même manière, sur ce graphe, nous ne pouvons pas identifier directement que le président sud-coréen est en Europe, pourtant on sait que le président est en visite à Paris qui est en France (grâce à la subsomption), qui est en Europe, nous devrions donc pouvoir déterminer le triplet <Président sud-coréen, est en visite en, Europe>.

Plus globalement, nous définirons les triplets, les subsomptions et les transitivités ainsi :

- **Triplets :** Notés ainsi : <Entité 1 ; Relation ; Entité 2>, ceux-ci représentent dans le graphe une liaison entre deux entités. Attention tout de même, la relation est orientée, c'est-à-dire que celle-ci va de l'entité 1 à l'entité 2. Ces triplets peuvent ensuite être représentés dans un « graphe de connaissances », soit un grand réseau d'entités (représentées comme les sommets du graphe) et de relations / propriétés (représentées comme les arcs du graphe). Dans la suite, un triplet pourra être noté $< s, p, o >$ où s pourra également être nommé « tête » ou « sujet », p « relation », « propriété » ou « prédicat » et o « queue » ou « objet ».

- **Subsomption :** Nous dirons dans la suite du rapport qu'il y a subsomption entre relations lorsque celle-ci peut être remplacée par une autre. Autrement dit, il y a subsomption lorsqu'un triplet $< A; \alpha; B > \Rightarrow < A; \beta; B >$. En explicitant cela, il est

donc important de mentionner que la subsomption est orientée, c'est-à-dire que :

$$(< A; \alpha; B > \Rightarrow < A; \beta; B >) \nLeftrightarrow (< A; \beta; B > \Rightarrow < A; \alpha; B >)$$

Nous désignerons ainsi, dans un premier temps, de subsomption unilatérale « par la gauche » quand α peut remplacer β, « par la droite » quand β peut remplacer α et de subsomption « bilatérale » (plus tard équivalence) quand nous pouvons remplacer α et β dans les deux sens.

- **Transitivité :** Ici, nous établirons, dans un premier temps, la transitivité comme étant la capacité d'une relation à en impliquer une autre, il s'agit donc aussi d'une opération orientée. Autrement dit, nous observerons qu'il existe une transitivité lorsque :

$$(< A; \alpha; B > \Rightarrow < B; \beta; C >) \Rightarrow < A; \gamma; C >$$

Ici, il faut également faire attention de ne pas confondre une transitivité avec un simple lien, par exemple : <Arno ; A écrit ; Rapport> et <Rapport ; à ; Mandelieu> n'implique pas nécessairement le triplet <Arno ; à ; Mandelieu> (la nuance sera expliquée plus en détail dans la partie concernant la transitivité).

Le but de ce stage est donc de faire un premier pas pour l'intégration de ces types de liens de subsomptions, de transitivités et plus globalement de connaissances implicites dans le modèle préexistant.

Maintenant que l'on a introduit le contexte dans lequel va se dérouler le stage et défini les termes les plus importants, nous pouvons commencer à analyser et comprendre les missions du stage. Mais avant, observons un peu le cadre existant.

C – Description du cadre RULK

Le cadre RULK (**R**epresenting **U**ser **L**earning and **K**nowledge) **[03]** est un Framework permettant de représenter les connaissances d'un utilisateur sur un sujet. L'idée de celui-ci part du principe que lorsque l'utilisateur recherche une information, il lit et comprend le contenu textuel des pages qu'il a cliqué, nous allons donc extraire les informations présentes sur cette page et les ajouter à ses connaissances. Afin de calculer le gain de connaissance engendré par l'ajout de cette information, nous prenons comme référence l'article Wikipédia du sujet associé à la recherche de l'utilisateur. Nous considérons donc dans ce cadre que l'objectif de l'utilisateur est d'obtenir au moins les connaissances présentes sur la page Wikipédia.

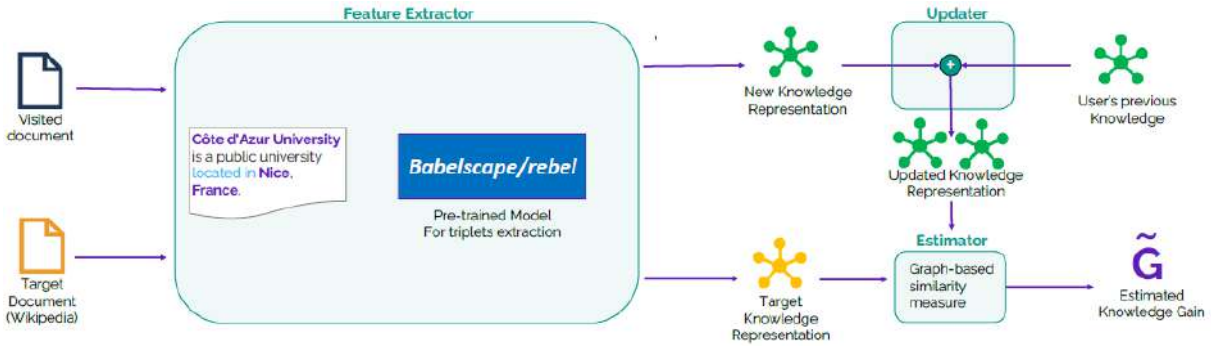
Si l'on rentre un peu plus en détail dans le fonctionnement de RULK, celui-ci dispose de trois composants principaux :

- **L'extracteur de connaissances (γ)**, qui agit comme le composant permettant l'extraction des connaissances de la référence et des pages visitées par l'utilisateur. Il en existe plusieurs implémentations permettant d'observer des facettes différentes des connaissances de l'utilisateur :

- **RULK$_{KW}$ [03]**, représentant les connaissances sous la forme d'un vecteur de m éléments où ces derniers représentent le nombre de fois que les m mots les plus fréquents sont apparus (stopwords exclus).
- **RULK$_{LM}$ [03]**, où BERT **[05]** est utilisé comme un extracteur de texte. La connaissance est ensuite représentée comme un vecteur dans un espace vectoriel à m dimensions à l'aide d'un embedding.
- **RULK$_{NE}$ [06]**, se basant sur les entités nommées, c'est-à-dire qu'il reconnait des entités (pays, organisations, etc.) et établit des liens entre elles. Les connaissances sont représentées à l'aide d'un vecteur contenant le nombre d'apparitions des 10 entités les plus fréquentes.
- **RULK$_{KG}$ [04]**, agissant comme une extension de RULK$_{NE}$, où les relations entre les entités sont aussi considérées, pour calculer le gain de connaissance d'un utilisateur. Les connaissances sont représentées sous la forme d'un graphe orienté (voir **I – A/B**).

- **Le metteur à jour (σ)**, qui permet à RULK de mettre à jour les connaissances de l'utilisateur au fur et à mesure de sa session d'apprentissage. Voici les différentes actions faites par celui-ci en fonction de la version de RULK :
  - **RULK$_{KW}$**, qui additionne simplement le vecteur des connaissances de l'utilisateur à celui des connaissances récupérées sur la page visitée (on additionne le nombre d'occurrences des mots).
  - **RULK$_{LM}$**, qui additionne encore une fois les vecteurs (ici des Embedding).
  - **RULK$_{NE}$**, additionne toujours les vecteurs des 10 entités les plus fréquentes.
  - **RULK$_{KG}$**, qui rajoute les connaissances de la page au graphe de l'utilisateur.

- **L'estimateur (θ)**, permettant au cadre de calculer le gain de connaissance de l'utilisateur. Cela se fait grâce à une mesure de similarité entre les connaissances ciblées et les connaissances actuelles de l'utilisateur. La mesure de similarité est la même pour toutes les instances de RULK (cosinus de similarité), excepté pour RULK$_{KG}$ qui dispose de sa propre mesure. Voici une représentation illustrée de RULK$_{KG}$ créée par M. Nasser :



Nous y prenons les documents consultés par l'utilisateur en plus de l'article Wikipédia cible, puis nous récupérons les triplets de ces derniers, ensuite, nous mettons à jour le graphe de connaissances de l'utilisateur, avant de le comparer au graphe des connaissances cibles afin d'estimer le gain de connaissances.

II – Missions du stage

Dans cette partie, nous allons, pour chaque mission confiée, réécrire mot pour mot ce qui est contenu dans le sujet de stage **[07]** et essayer de les développer pour en comprendre les implications.

Première mission : « Construire une base de connaissances avec toutes les inférences possibles permettant de prendre en compte, entre autres, la transitivité des relations, comme c'est le cas de la relation "est-en" dans l'exemple précédent, ou la subsomption des relations, comme c'est le cas de la relation "est-la-capitale-de" qui implique "est-en". »

Ici, il faut donc créer un graphe prenant en compte les types de relations précédemment cités. Les principales difficultés liées à cette mission sont de bien définir les subsomptions, mais aussi d'éviter la confusion entre un lien simple et une transitivité comme expliqué précédemment. Les outils et compétences utilisés pour cette mission relèveront sans doute principalement de la programmation en Python, de la synthèse d'informations, mais aussi dans une moindre mesure de l'utilisation d'ontologies / graphes de référence pour l'extension de graphes (IA symbolique).

Deuxième mission : « Prendre en main le code existant écrit en Python. »

Pour cette mission, il s'agit de comprendre le code déjà écrit afin de pouvoir, par la suite, y intégrer les ajouts effectués lors de la première mission.

Les compétences développées ici reposeront probablement encore une fois sur Python, mais plus spécifiquement sur l'utilisation de librairies pour le traitement du langage naturel (NLP) ou de Jupyter Notebook.

Troisième mission : « Intégrer la nouvelle base de connaissances dans le cadre existant. »

Il s'agit de la suite logique de la première mission et de la deuxième mission, soit d'appliquer ce qui a été créé précédemment.

Quatrième mission : « Réaliser des expériences pour vérifier la valeur ajoutée de ce nouveau cadre par rapport à la littérature. »

Concernant cette mission, l'objectif est de comparer ce qui a été fait précédemment, à ce qui a été apporté, cela se fera à l'aide de métriques que nous détaillerons plus tard.

Les compétences utilisées ici relèveront surtout de l'analyse des résultats. L'objectif étant de comprendre si nous observons une amélioration significative du cadre initial grâce à l'utilisation de connaissances implicites.

Cinquième mission : « Éventuellement participer à la rédaction d'un article scientifique faisant état des résultats obtenus. »

Enfin, si les résultats sont concluants et que le temps le permet sur la période de stage, il faudra résumer les résultats au sein d'un article scientifique afin d'y publier les contributions.

Globalement, ces missions semblent assez intéressantes et devraient me permettre de mieux comprendre l'univers de la recherche en participant à certaines phases importantes de celle-ci : la recherche d'informations, le développement et la restitution des résultats.

Pour ces missions, il m'est fourni initialement huit articles scientifiques **[07]** pour comprendre le sujet du stage et le domaine d'études. Ces articles en incluent trois au format papier, concernant directement le cadre RULK à étendre lors du stage. Il reste néanmoins possible de s'approvisionner en article grâce à Google Scholar notamment.

M'est également fourni l'ensemble du code Python, accompagné de sa documentation, me permettant de mettre en place le cadre RULK et de quantifier l'amélioration de la représentation du gain de connaissances de l'utilisateur.

III – Présentation du laboratoire

A – Introduction

Le laboratoire I3S/CNRS **[08]** (Informatique, Signaux et Systèmes de Sophia Antipolis) est un laboratoire de recherche public créé en 1989 suite à la fusion de deux laboratoires (LASSY et LISAN).

Celui-ci est placé sous tutelle du CNRS **[09]** (Centre National de Recherche Scientifique, rattaché à l'INS2I **[10]** : Institut des Sciences de l'Information et de leurs Interactions), de l'université Côte d'Azur **[11]** et plus secondairement de l'INRIA **[12]**. Il s'agit donc d'une unité mixte de recherche **[67]** (UMR 7271).

Le laboratoire est situé sur le campus de SophiaTech et est composé de près de 300 membres, principalement des enseignants-chercheurs et des doctorants complétés par des stagiaires de master ou d'écoles d'ingénieur.

B – Organisation

Le laboratoire I3S est une organisation fonctionnant autour de quatre pôles (COMRED, MDSC, SIS et SPARKS), ces pôles disposent d'axes de recherche distincts les uns des autres, mais restent principalement dans le domaine de l'informatique.

- **COMRED** (Communications, Network, Embeded and Distributed Systems) : Ce pôle se charge des problématiques de conceptions avancées basées sur des modèles formels dans les domaines des systèmes embarqués, distribués et des réseaux de communications.
  - **Équipes :** EPC COATI, EPC Kairos, équipe Scale

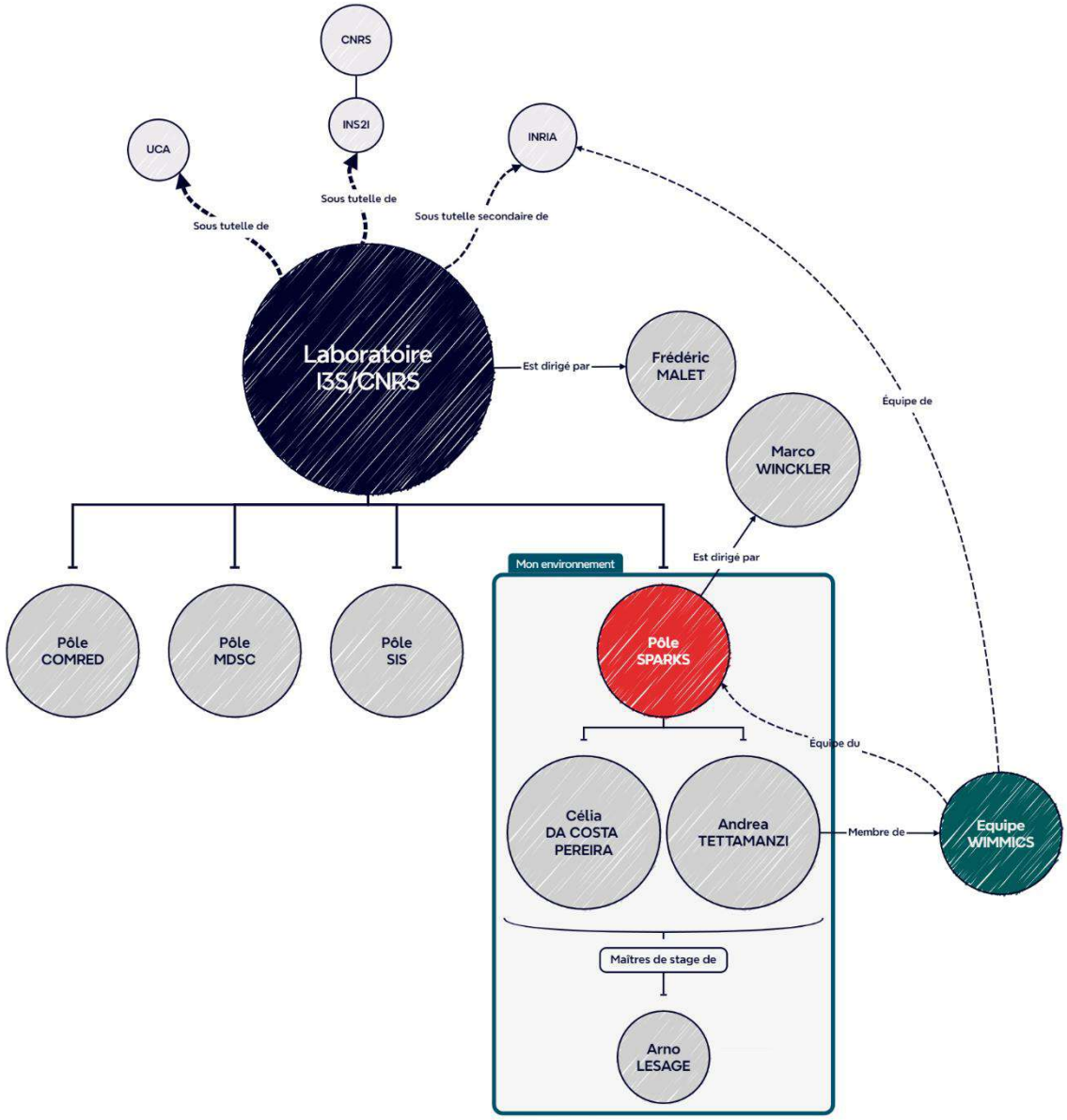
- **MDSC** (Modèles Discrets pour les Systèmes Complexe) : Ce pôle se charge des modèles formels pour les systèmes complexes et de leurs applications dans différents domaines (principalement dans la Biologie, les Mathématiques et l'Informatique). L'équipe présente aussi une forte activité dans le domaine de la programmation par contraintes et de la vérification de programmes.
  - **Équipes :** C&A, MC3

- **SIS** (Signaux, Images et Systèmes) : Dans celui-ci, les recherches menées se font, entre autres, autour du traitement du signal, des images, de la compression de données multimédias, mais aussi autour de la robotique et des systèmes autonomes.
  - **Équipes :** Équipe Signal, EPC Morpheme, équipe Mediacoding, équipe OSCAR, équipe Robot Vision, équipe Hybride, équipe Fault-IA, équipe Design

- **SPARKS** (Scalable and Pervasive Software and Knowledge Systems) : Il s'agit du pôle dans lequel je suis intégré. Au sein de celui-ci, les axes de recherche sont plutôt orientés vers l'extraction des connaissances, l'apprentissage automatique, l'analyse des interactions à travers les graphes, le web sémantique ou encore sur l'informatique pour la biologie.

- **Équipes :** WIMMICS, MAASAI

Voici un organigramme simplifié de l'organisation :



C – Informations diverses

Pour commencer, voici des informations en vrac par rapport au laboratoire I3S et de ses établissements de tutelle : le CNRS, l'université Côte d'Azur et l'INRIA **[13, 14]** :

Infos.	Laboratoire I3S	CNRS	UCA	INRIA
Effectif	300	> 33 000 (2022)	> 3000	4742 (2022)
SIRET	18008901306558	18008901303720	13002566100013	18008904700047 (Côte d'Azur)
Code APE	7219Z	7219Z	8542Z	7219Z
Budget	N/A	3 801 M€ (2022)	291.6 M€ (2022)	334 M€ (2023)

Ensuite, concernant ce qui est produit par le laboratoire, cela concerne principalement des publications d'articles scientifiques, rendues disponibles pour la plupart sur HAL **[15]**. Cette activité implique donc nécessairement de la rigueur, où chacune de nos actions sur un projet doit pouvoir être expliquée de manière claire à un Reviewer, soit une personne qui relie les articles avant que ceux-ci soient publiés. Dans ces « produits » nous avons aussi des logiciels principalement à destination de la recherche **[16]**, mais aussi des livres et dans une moindre mesure des start-ups.

IV – Accomplissement des missions du stage

Dans cette longue partie, nous discuterons des actions effectuées afin de compléter les missions confiées, l'organisation que j'ai faite ainsi que les problèmes rencontrés avec leurs solutions.

A – Identification des problèmes

Pour commencer, la première semaine n'a pas été dédiée à la recherche de solutions particulières pour chacune des missions, mais plutôt à la compréhension du sujet, du domaine et à l'identification des problèmes à résoudre. J'ai donc lu la documentation et analysé le code préexistant et voici certains problèmes que j'ai pu identifier ou retrouver par rapport au sujet du stage :

1. **Une notion peut disposer de plusieurs entités pourtant identiques d'un point de vue sémantique.**

   Par exemple, si nous avons pour cible la notion « Collateralized Debt Obligation » **[17]**, en fonction du texte traité, nous retrouverons dans le graphe : « CDO », « CDOs », « CDO's » ou « Collateralized Debt Obligation ». Cela pose problème, car nous diluons l'information que l'utilisateur pourrait connaître. Ainsi si nous avons une liaison stipulant que « CDO » est une sous classe de « MBS » et une autre disant que « CDOs » est une sous classe de « debt instrument », nous ne pouvons pas savoir que l'utilisateur sait que « CDO » est une sous classe de « debt instrument », même si « CDO » et « CDOs » sont les mêmes choses sémantiquement parlant.

   Il faut donc trouver un moyen de les considérer comme une unique entité, une solution serait de mettre en place un prétraitement (retirer les apostrophes s, etc.) et un système de stemming, soit une méthode permettant d'obtenir la racine d'un mot. Enfin, pour le dernier cas (la signification de l'acronyme), il faudra essayer d'être plus imaginatif.

2. **Les liens de transitivité et de subsomption ne sont pas en place.**

   Imaginons l'exemple suivant : « No Documentation Mortgage » est une sous-classe[1] de « Adjustable Rate Mortgage » et que « Adjustable Rate Mortgage » est un sous-type de « Mortgage », ici on se doute bien par transitivité que « No documentation Mortgage » est un sous-type ou une sous-classe de « Mortgage ». Malheureusement, le graphe ne le sait pas pour deux raisons, la première est que celui-ci ne sait pas ce qu'est la transitivité et la deuxième est que ce dernier ne sait pas qu'un sous-type est une sous-classe.

[1] Il n'est pas rigoureux de mélanger sous-type et sous-classe dans la réalité, mais cela paraît être une bonne chose d'un point de vue explicatif.

Pour créer les triplets, nous utilisons un modèle permettant d'en générer depuis un texte : REBEL **[18]**. L'ensemble des relations pouvant être générées par le modèle est connu, on pourra donc essayer de regarder à la main quelles relations peuvent être remplacées par d'autres et quelles relations impliquent une transitivité.

3. **Un graphe peut ne pas être complet à cause d'une connaissance de « base » non mentionnée dans le texte traité.**

   Un exemple simple serait de dire que le chiot est l'enfant du chien qui est un animal et que le chaton est l'enfant du chat. Ainsi, avec le texte, nous savons que le chien est un animal, mais nous ne savons pas que le chat en est un. Cela relève d'une connaissance de « base », notion compliquée à définir étant donné que cette « base » dépend de chacun. Ainsi, si nous considérions que « le chat est un animal » constitue une connaissance de base, nous devrions être en mesure de connaître cela, chose que le graphe ne sait pas.

   Une solution à ce problème pourrait se trouver dans les graphes de référence, avec lesquels nous pourrions étendre les connaissances de l'utilisateur si nous arrivions à distinguer les connaissances de « base » dans celui-ci. Un graphe de référence que l'on pourrait utiliser pour cette tâche pourrait-être DBpedia **[19]**.

   Une autre solution serait d'utiliser un dictionnaire d'ontologie **[20]**, le problème avec cette méthode provient de la fréquente spécialisation de ces dictionnaires dans un domaine précis, néanmoins, ces derniers sont plus « simples », car ils sont créés pour servir de base.

À partir de là, nous avons donc déjà trouvé quelques idées que l'on pourrait appliquer pour que le graphe des connaissances de l'utilisateur prenne en compte les connaissances implicites.

B – Prise en main du code existant et base de connaissances

B.1 – Compréhension des concepts

Avant toute chose, le plus important pour construire une base de connaissances prenant en compte la subsomption et la transitivité est de savoir si nous comprenons bien la documentation, les concepts sur lesquels elle se base et enfin si nous arrivons à en produire un code indépendamment de ce qui a été créé.

Pour cela, la première chose qui a été faite est d'extraire les triplets du texte, étant donné que l'objectif de cette partie n'est pas de constituer des résultats, que nous avons déjà, mais juste de comprendre le code, nous pouvons nous contenter d'extraire les triplets depuis les sept articles Wikipédia seulement, qui constituent les représentations des connaissances que souhaitent atteindre les utilisateurs.

Les articles Wikipédia sont stockés dans le fichier « wikipedia_texts.tsv » (le format TSV veut dire Tabulation-separated values, il est assez similaire au format CSV), nous avons dans celui-ci, en première colonne, le nom de l'article et en deuxième colonne, le texte contenu dans l'article. Voici, à quoi l'intérieur du fichier ressemble :

```
Subprime%20mortgage%20crisis	Author Michael Lewis wrote that CDS […]
Irritable%20bowel%20syndrome	The use of opioids is controversial […]
Radiocarbon%20dating%20considerations	In the early years of using […]
[…]
```

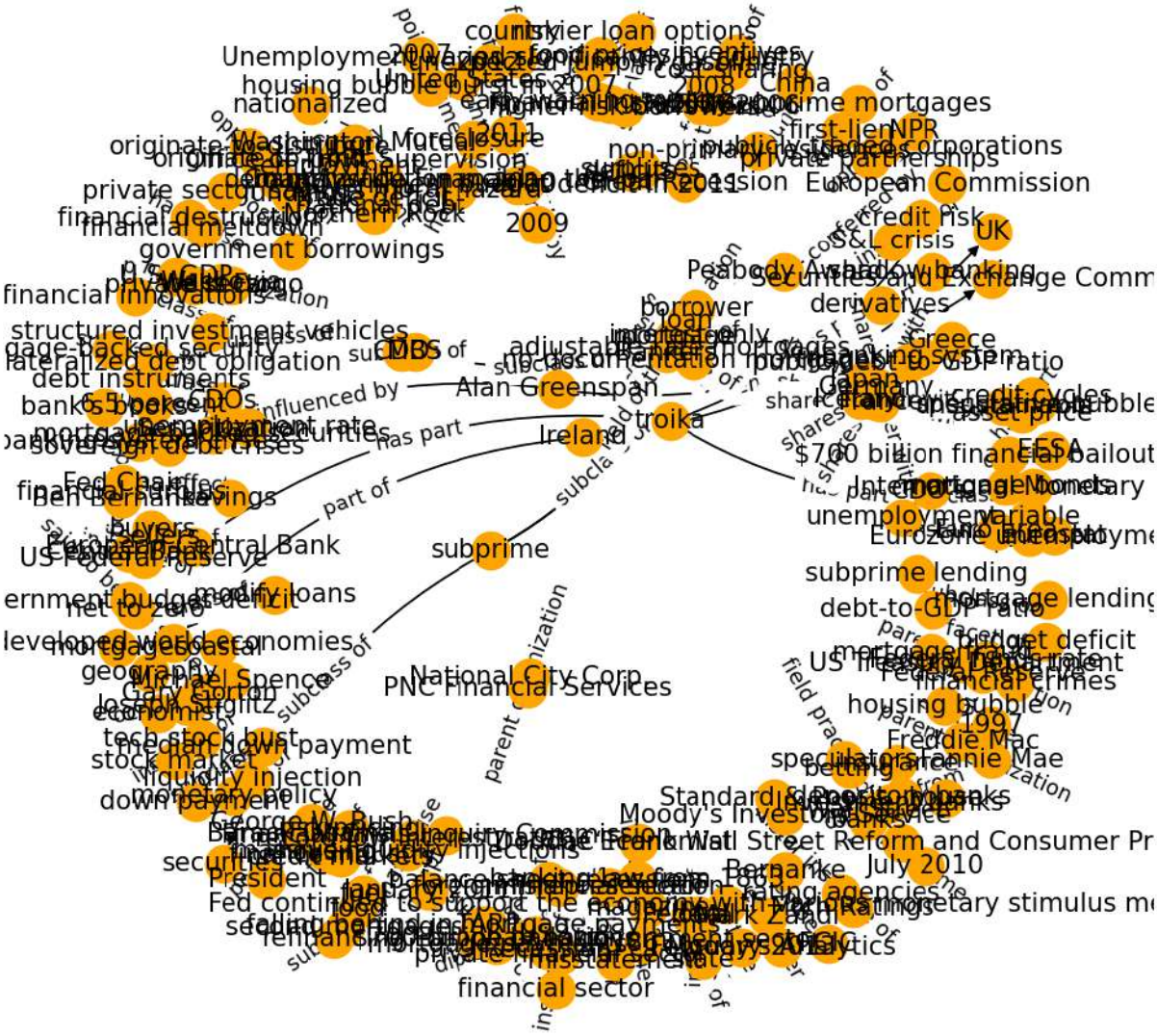
L'extraction du texte est donc extrêmement simple, il suffit de lire le fichier ligne par ligne et de récupérer le contenu de chaque article. Ensuite, il suffit d'exploiter REBEL, déjà utilisé dans le code de base, le modèle permet d'extraire des triplets via un texte (voir **[21]** pour l'article détaillant son fonctionnement). Cela nous permet déjà d'avoir un graphe. Voici les étapes simplifiées du pseudo-algorithme que j'ai utilisé :

```
Modules nécessaires dans Python : Spacy, Pandas
Fichier du code Python équivalent : rebelSpacy.py

Début Algo extraireTriplets :
    var Contenu (liste) ← [ ]
    var Données[titre, contenu] (dataframe) ← dataframe()
    Pour chaque ligne dans « wikipedia_texts.tsv » :
        Contenu ← dernier élément de ligne.split("\t")
        Données ← Données + [premier élément de ligne.split("\t"),REBEL::extract_triplets(Contenu)]
    Fin Boucle
    Sauvegarder Données[titre, contenu] au format CSV
Fin Algo extraitreTriplets
```

Nous savons maintenant comment se servir de REBEL dans Python pour extraire des triplets. Ceux-là constituent déjà un graphe d'une certaine manière, mais nous ne pouvons pas vraiment le « voir », or un graphe est aussi un outil qui permet d'avoir un visuel sur des relations de manière simple, nous allons donc créer un bout de code pour pouvoir l'afficher (celui-ci sera présent dans le fichier plotKG.py[2]), pour cela nous utilisons dans Python les modules « NetworkX » **[68]** et « matplotlib » pour afficher les graphes. Voici ce que cela donne pour le sujet « Crise des Subprimes » :

[2] Le fichier a fini par être supprimé dans l'avancement du stage, mais le code permettant l'affichage des graphes a été conservé dans la classe KB du script Python KnowledgeBase.py sous la fonction KB.plotGraphe.



Remarque : Même si ce graphe semble être assez complexe, il ne l'est en réalité pas tant que ça par rapport aux autres documents qui constituent les connaissances des utilisateurs (j'entends ici par complexité, le nombre de sommets du graphe, le nombre d'arcs et la longueur des chaînes de relations). Sachez également qu'il est possible dans matplotlib de zoomer et de se déplacer dans le graphe, facilitant ainsi sa compréhension. C'est grâce à cela que les problématiques du **IV.A** ont été soulevés.

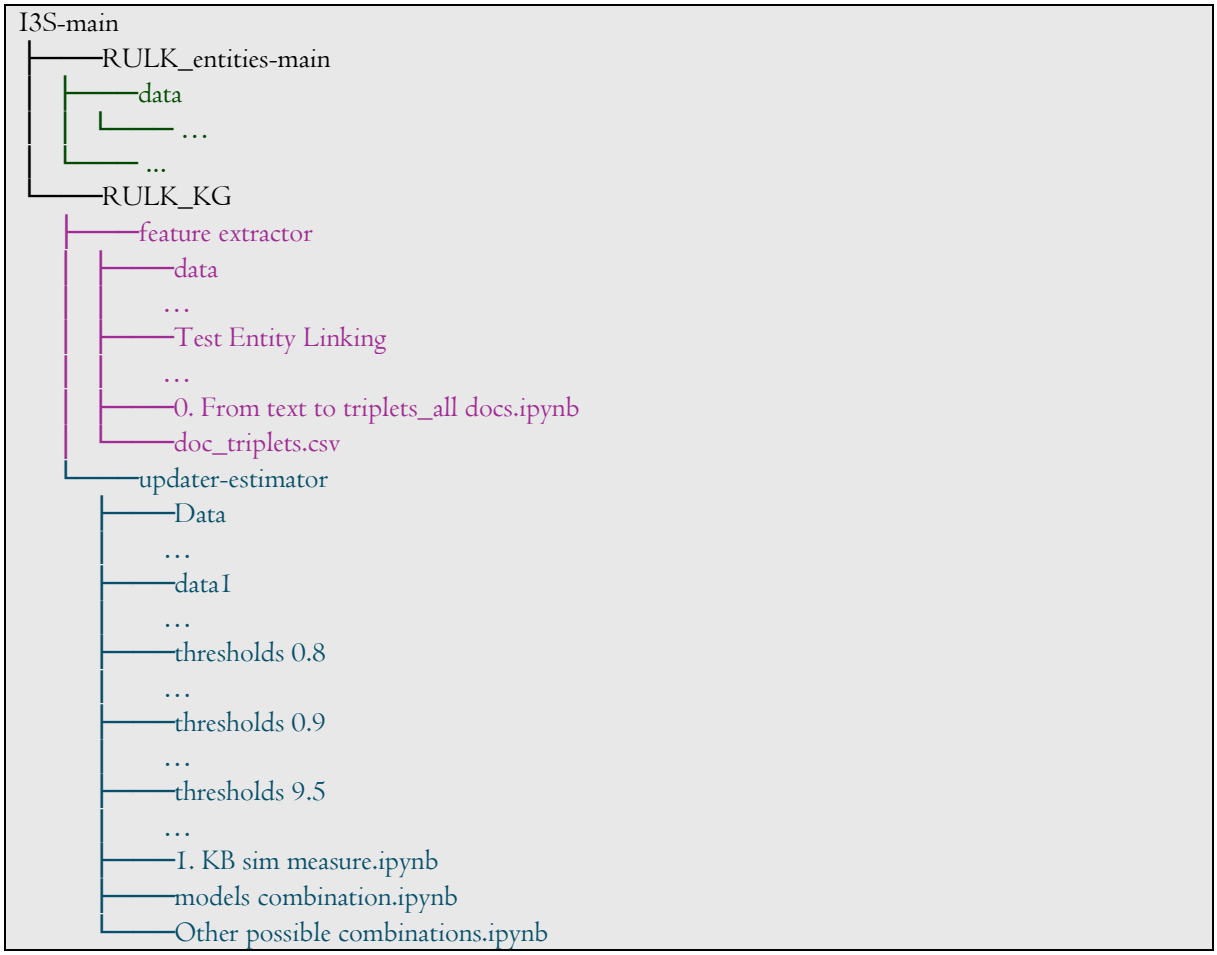
Attention : Pour générer le graphe, nous avons utilisé la librairie JSON de Python pour récupérer les relations. Il a donc fallu changer le format de ces dernières vers un format JSON valide. Cela s'est fait avec le code dans le fichier `tripletTransformer.py`[3].

Maintenant que nous avons compris comment nous servir des graphes dans Python et comment manipuler les triplets que nous retrouverons dans le code initial, nous pouvons nous intéresser à ce dernier.

[3] Le fichier a définitivement été supprimé. En effet, au fur et à mesure de l'avancement du projet, il n'était plus nécessaire de faire le graphe à partir d'un fichier JSON, bien que cela reste possible avec les paramètres de `KB.plotGraphe`.

B.2 – Analyse du code existant

Pour commencer, le code m'a été transmis en début de stage par M. Nasser via GitHub. Dans celui-ci, une structure particulière que voici est à respecter :



En analysant l'arborescence, nous pouvons déjà déduire certaines informations. Ainsi, en dessous de `RULK_entities-main` se trouve l'ensemble du code de la version de $RULK_{NE}$ se basant sur les entités nommées, cela ne nous intéressera pas dans ce projet.

Ensuite, nous avons la sous-arborescence de `RULK_KG`, qui comme son nom l'indique correspond au code de $RULK_{KG}$. Au sein de celle-ci, nous retrouvons le dossier `feature extractor`, correspondant à l'ensemble du code permettant l'extraction des triplets, nous y retrouvons donc naturellement dans son sous-dossier `data` des fichiers que nous utiliserons par la suite : `wikipedia_texts.tsv` contenant les triplets des articles Wikipédia ou `logs_with_position.json` contenant l'ensemble des documents consultés par chaque utilisateur, son gain de connaissance réel à la fin de l'expérience, son sujet d'apprentissage et son identifiant.

Enfin, le dossier `updater-estimator` contient l'ensemble du code permettant l'évaluation des modèles, de manière simple ou en les combinant. Dans le sous-dossier `Data`, nous retrouvons certains fichiers précédents auxquels se rajoutent les fichiers : `clicked_docs_with_topics.tsv` contenant pour chaque document consulté, le lien internet du

document, l'article Wikipédia associé et le contenu textuel du document ou encore `details_trpilets.csv` contenant pour chaque utilisateur, leur identifiant, le sujet qui les concerne, les triplets de l'article Wikipédia associé et les triplets que l'utilisateur a cumulés lors de ces recherches.

Nous connaissons maintenant l'architecture du code. Observons donc les fichiers qui nous intéresseront directement dans la mise en place de nos missions :

I3S-main\RULK_KG\feature extractor\0. From text to triplets_all docs.ipynb :

Ce Jupyter Notebook permet l'extraction des triplets, mais aussi la génération et le maintien des graphes via une classe définie à l'intérieur **KB (Knowledge Base)**, cette classe est très importante, car à travers celle-ci sera gérée la subsomption et la transitivité.

L'extraction des triplets à partir d'un texte se fait dans le notebook à l'aide de la fonction `from_text_to_KB()` (il s'agit de l'extracteur de connaissances γ), elle renvoie un objet de la classe KB, cette classe est celle qui permet le maintien du graphe de connaissance de l'utilisateur en permettant notamment l'ajout de relations de manière simple.

C'est donc avec ce fichier que sont générés les graphes pour chaque document présent dans l'étude. Attention tout de même, le temps d'exécution est **très élevé** (comptez plus d'une semaine pour générer les graphes).

I3S-main\RULK_KG\updater-estimator\1. KB sim measure.ipynb :

Ce fichier est également important, car il permet de mettre en place l'évaluation du modèle. Plus précisément, il calcule la similarité entre la cible et les connaissances de l'utilisateur, mais aussi de mesurer la corrélation de cette connaissance estimée par rapport à la réalité. En somme, ce fichier assure le rôle du metteur à jour (σ), mais aussi de l'estimateur (θ).

I3S-main\RULK_KG\updater-estimator\models combination.ipynb :

Enfin, celui-ci est le dernier fichier qui nous intéressera réellement d'un point de vue code. En effet, il nous permettra lors de la phase d'expérimentation de voir quels seraient les meilleurs modèles à combiner pour représenter la connaissance d'un utilisateur.

Globalement, nous observons que le code n'est pas excessivement complexe, néanmoins, en raison du manque de commentaires faits sur celui-ci, il est compliqué de s'y retrouver au début. De plus, l'utilisation de Jupyter Notebook, rend la gestion des classes et des fonctions plus floues, j'ai donc décidé une fois le code compris de le transvaser vers une version orientée objet dans plusieurs fichiers Python séparés pour augmenter la lisibilité. En voici la nouvelle arborescence[4] :

[4] Cette arborescence n'est plus vraiment celle du projet final, mais reste néanmoins suffisamment similaire pour s'y retrouver de manière simple.

```
Code
    KnowledgeBase.py
    playgroundCode.py
    Similarity Mesure.ipynb
    SimilarityCalculator.py
    UserBase.py
├───BackUp
│       …
├───baseCode
│       …
├───Data
│       clicked_docs_with_topics.tsv
│       details_trpilets.csv
│       doc_triplets.csv
│       logs_with_position.json
│       subsomptionRelation.json
│       transitivityRelation.json
│       wikipedia_texts.tsv
│       wiki_triplets.csv
├───exportKB
├───exportKnowledgeExtractor
├───exportUB
└───Result
```

Ce qu'il faut retenir de cette architecture, c'est que la classe KB se situe maintenant dans KnowledgeBase.py et que l'évaluation se fait via la classe SC (Similarity Calculator) du fichier SimilarityCalculator.py. Les données utilisées sont dorénavant présentes dans le sous-dossier Data et les résultats dans les sous-dossiers export…[5] ou Result pour l'évaluation des modèles.

B.3 – Création de la base de connaissances

Comme implicité plus tôt, nous allons utiliser la classe KB pour créer la base de connaissances. Dans celle-ci, il existe de base sept fonctions et deux variables d'instance self.entities et self.relations. La première correspond à un dictionnaire contenant en clé le nom de l'entité et en valeur son URL, si elle en dispose et un résumé/définition si elle en possède. La deuxième variable est bien plus intéressante. En effet, il s'agit de cette variable qui stocke réellement le graphe. Ce dernier se stocke sous ce format :

```
Format :
[{"head":"…", "type":"…", "tail":"…"}, {…}, …]
Example :
[
    {"head":"Antibes, "type":"located in the administrative territorial entity", "tail":"CASA"},
    {"head":"CASA", "type":"area", "tail":"Southeast France"},
    {"head":"I3S", "type":"part of", "tail":"CNRS"},
    {"head":"I3S", "type":"part of", "tail":"Université Côte d'Azur"},
    {"head":"I3S", "type":"part of", "tail":"INRIA"}
]
```

[5] Le dossier n'est plus exploité dans la version définitive, mais reste disponible dans le sous dossier Bonus.

Ce sera donc à partir de cette variable que nous inclurons la subsomption et la transitivité. Concernant la fusion des entités au sens sémantique identique, il faut agir au moment de l'extraction des relations[6]. Nous allons donc, avant de toucher aux relations, mettre en place le système permettant la fusion de ces entités.

B.3.a – Création de la base de connaissances, fusion des entités

Pour fusionner les entités, nous utilisons une combinaison de procédés se basant autour du stemming (un système qui permet de retirer les suffixes d'un mot pour le transformer en un mot racine).

Si nous reprenons l'exemple précédent utilisant les « CDO », nous avons dans le graphe cible traitant des Subprimes les entités suivantes : « Collateralized Debt Obligation », « CDO », « CDOs » et « CDO's ». En appliquant le stemming en Python, en utilisant le module « nltk » (Natural language toolkit **[22]**), nous observons que « CDOs » fusionne bien avec CDO, mais ce n'est pas le cas des autres. C'est à ce moment-là que nous devons être plus imaginatifs, comprendre comment l'on exploite le stemming et plus précisément pourquoi « CDO's » ne s'est pas fusionné avec « CDO ».

Avant d'utiliser le stemming sur une phrase, nous devons créer des « tokens » **[23]**, il s'agit pour faire simple de séparer notre phrase en une liste de plusieurs lettres/mots sur lesquelles pourra travailler notre programme. En utilisant encore une fois « nltk », « CDO » reste « CDO », « CDOs » reste « CDOs », « Collateralized Debt Obligation » devient [« Collateralized », « Debt », « Obligation »], mais « CDO's » devient [« CDO », « 's »]. Or, en appliquant le stemming à chaque élément de la liste séparément avant de la reconcaténer, le « 's » reste pour « CDO's ».

Pour résoudre cela, il suffit donc de prétraiter la phrase afin que « 's », ne soit pas traité comme un token séparé. Ainsi, avant toute opération de stemming sur le texte, nous appliquons les opérations suivantes :

1. Le **texte** est mis en **minuscule** pour que « MBS » soit considéré de la même manière que « Mbs » ou « mbs »,
2. On **retire** les **guillemets**, ainsi si nous prenons « "troïka" », celui-ci deviendra « troïka »,
3. On **retire** les **« 's »**, ainsi « Moody's » devient « Moody »,
4. On **retire** les **« ' »**, ainsi « AAA' » rebascule en « AAA ».

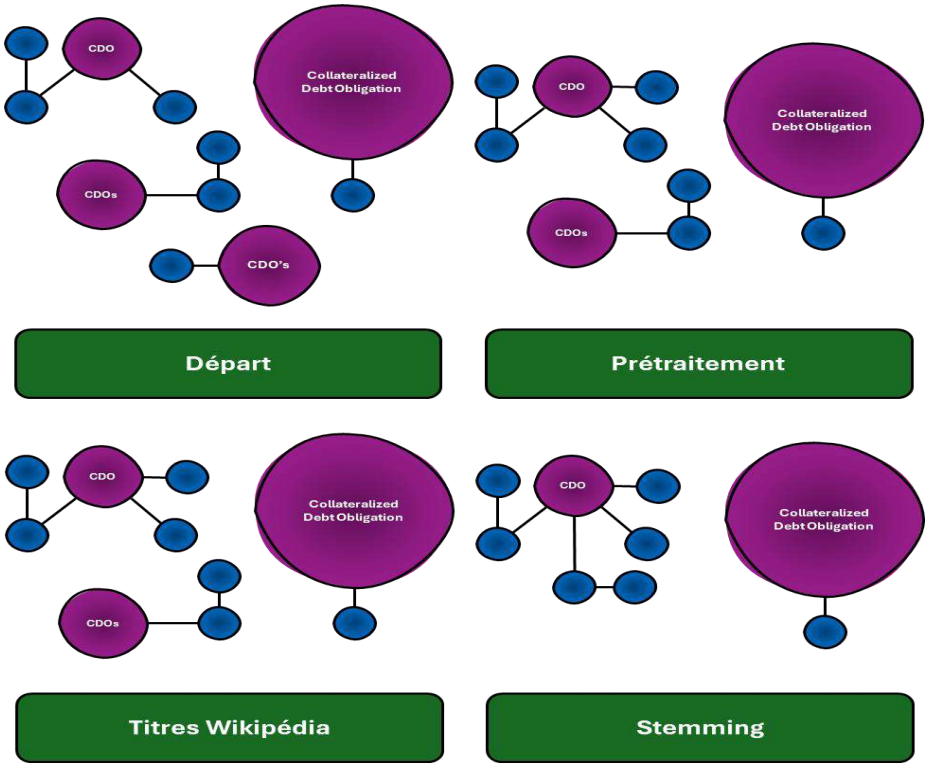
En appliquant les étapes précédentes, nous observons dorénavant que « CDO », « CDOs » et « CDO's » sont bien combinés en « CDO », ce n'est cependant pas le cas du dernier, qui ne peut pas être fusionné par ce genre de méthode. Une solution à laquelle j'avais pensé était de se rabattre sur des dictionnaires d'acronymes, le problème est qu'un acronyme peut avoir plusieurs significations, ainsi « CDO » veut bien dire « Collateralized Debt Obligation » lorsque l'on est en économie ou sur les marchés financiers, mais désigne aussi « Continuous Descent Operations » une opération

[6] Il a été soulevé à la fin du stage que nous aurions pu effectuer cette étape sur les triplets existants dans le graphe sans les réextraire du texte.

d'optimisation des profils des vols en aéronautique et « Chief Data Officer » une fonction de direction dans une entreprise liée aux données **[24]**. Il faudrait donc avoir un dictionnaire différent pour chaque cible, ce qui n'est pas toujours possible (car sur certains sujets précis ou peu exploités, il n'existe pas nécessairement de dictionnaires ciblés sur le dit sujet pour réduire le nombre d'acronymes à vérifier[7]).

Une solution partielle trouvée par M. Nasser précédemment est d'utiliser le fonctionnement des titres des articles Wikipédia. Par exemple, si avec le module Python `wikipedia`, nous tapions `wikipedia.page("Napoléon Bonaparte")`, nous serions redirigés vers la page « Napoleon », cela nous permet de fusionner les entités « Napoléon Bonaparte » et « Napoléon » en « Napoleon ». Néanmoins, dans l'implémentation initiale faite par M. Nasser, si la page n'est pas trouvée, nous en supprimions l'entité, ce qui entraîne inévitablement une perte conséquente d'informations. Nous avons donc considéré qu'il serait préférable de ne pas les retirer, quitte à avoir deux entités différentes avec un sens sémantique identique.

De plus, cette solution ne règle malheureusement pas notre problème de « CDO » et de « Collateralized Debt Obligation », étant donné que la page « CDO » peut se référer à plusieurs définitions (ce qui empêche la fusion des entités). La solution reste néanmoins convenable dans certaines situations comme dans celles des acronymes à définition unique et des entités similaires à l'exemple de Napoléon au-dessus. Voici donc le résumé du traitement pour la fusion des entités :



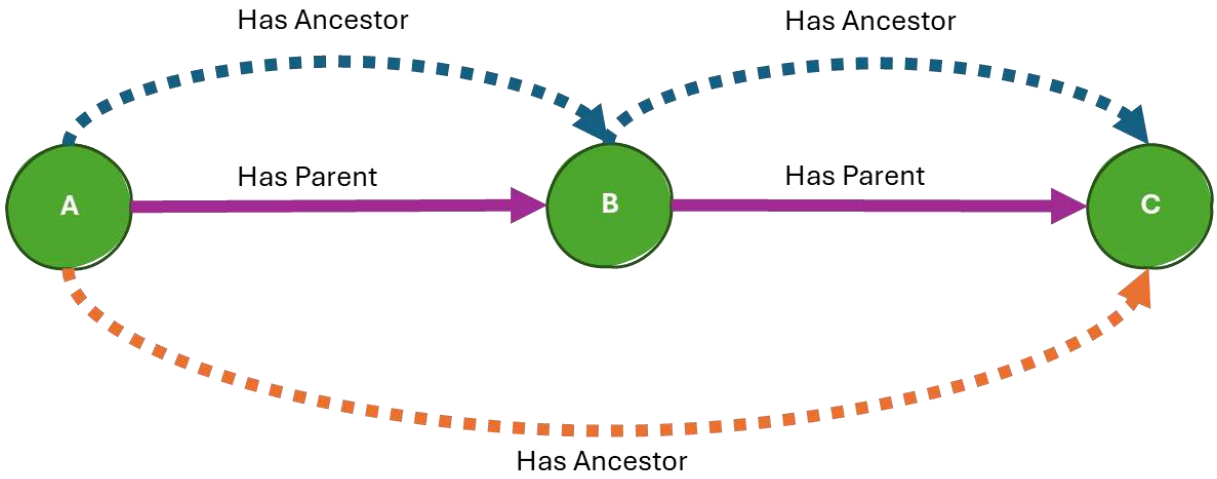
[7] Il existe bien des dictionnaires d'acronymes très complets (comme https://www.les-abreviations.com/repertoires-sigles.html), mais nous cherchons avant tout un dictionnaire plus petit sur un sujet, afin d'en trouver la bonne définition dans un contexte précis. De plus, même s'ils existaient, ils ne sont pas centralisés, il faudrait donc effectuer des recherches pour en trouver un nouveau à chaque fois et cela ne permet pas une mise à l'échelle simple, car l'action n'est pas automatique.

B.3.b – Création de la base de connaissances, relations de subsomption

Maintenant que nous avons réussi à mettre en place le prétraitement, nous pouvons nous pencher sur une partie de la mission principale du stage : mettre en place la subsomption des relations (voir **I – B** pour définition). Mais pourquoi commencer par la subsomption plutôt que par la transitivité ?

Pour commencer, il faut bien comprendre que la plupart des opérations que l'on effectue sur les graphes ne sont pas commutatives. Ainsi, si nous effectuons en premier lieu la subsomption puis la transitivité, alors nous n'aurions pas le même résultat qu'en appliquant la transitivité puis la subsomption. Ainsi, l'idée initiale de commencer par la subsomption consistait à subsumer des relations par leur contrepartie transitives avant d'appliquer la transitivité.

Voici un exemple :



Ici, nous avons en violet les relations initiales « Has Parent », néanmoins, celle-ci n'est pas transitive (On peut dire que « A » a pour parent « B », mais on voit que « A » n'a pas pour parent « C » et ce même si « B » a pour parent « C »). Or, nous pouvons considérer qu'un parent est un ancêtre, on subsume donc le lien « Has Parent » par « Has Ancestor ». De plus, le lien « Has Ancestor » est transitif. En effet, si « A » a pour ancêtre « B » et « B » a pour ancêtre « C », alors nous pouvons dire que « A » a pour ancêtre « C » (lien orange). Si nous l'avions fait dans l'autre sens (transitivité puis subsomption), nous n'aurions trouvé que les liens en bleu, mais pas le lien en orange.

Note : Ici, nous pourrions aussi créer des liens dits « inverses » tels que « B Has Child A » ou « B Has Descendant A ». Si nous rajoutions le triplet « B Has Spouse D », alors nous pourrions aussi rajouter le lien symétrique « D Has Spouse B », « B » et « D » étant en couple. Nous reparlerons de ces autres liens dans la partie sur le web sémantique, RDF, RDFS et OWL.

Ainsi, pour mettre en place la subsomption, nous devons déjà savoir quels sont les liens à subsumer et par conséquent, mettre la main sur la liste exhaustive des relations pouvant être générées par REBEL. Nous trouvons cela sur le GitHub de REBEL **[18]**, dans `rebel/data/relations_count.tsv`. À partir de là, il « suffit » d'en extraire les relations et de

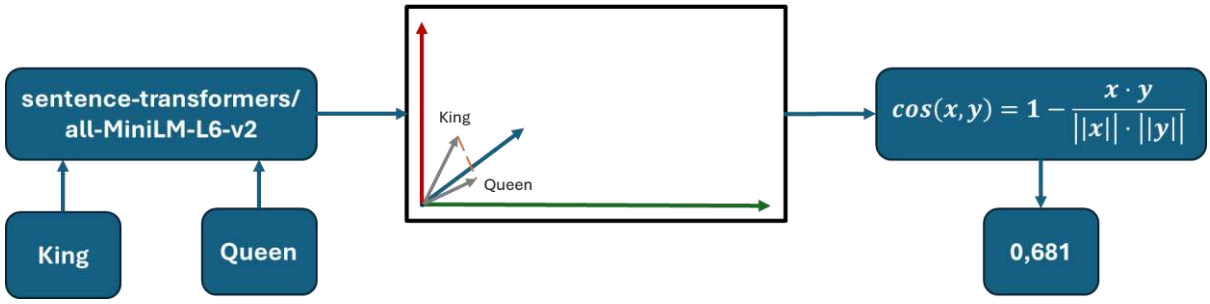
regarder l'ensemble des combinaisons possibles à la main. Or, il suffit de réfléchir rapidement pour comprendre que faire cela à la main ne sera pas forcément aisé et ce pour plusieurs raisons :

Premièrement, nous devons comprendre la sémantique des relations de manière approfondie pour chacune des décisions pour éviter les erreurs dues à des subtilités sémantiques. Par exemple les liens « IndigenousTo » et « EndemicTo » sont similaires dans la vie courante et pourraient s'utiliser interchangeablement, mais sémantiquement parlant, les deux relations revêtent des sens différents. En effet, la différence entre les deux est qu'une espèce endémique dispose d'un territoire défini et exclusif (dans le sens où elle n'en possède pas ailleurs), alors qu'une espèce indigène peut avoir plusieurs territoires dans plusieurs régions **[25]**. La relation sémantique qui lie les deux relations est donc « EndemicTo » sous-relation de « IndigenousTo ».

Deuxièmement, il suffit d'un calcul simple pour comprendre qu'il y a beaucoup de relations à vérifier. En effet, en considérant qu'une subsomption ne peut s'effectuer qu'entre 2 relations et que nous avons 230 relations gérées par REBEL, il nous faudra en réalité vérifier $\binom{230}{2}$ combinaisons, soit 26 335 relations de subsomption possibles. Cela rend la comparaison à la main pour l'ensemble des relations trop chronophage pour la durée du stage (3 mois).

La première problématique n'est pas automatisable de manière simple, néanmoins la seconde l'est partiellement.

Ainsi, pour réduire le nombre de possibilités, nous utilisons un modèle de langage pré-entraîné afin d'évaluer la ressemblance sémantique des relations de manière quantitative et de présélectionner les relations de subsomption parmi les couples qui dépassent un certain seuil. Ici nous utiliserons le modèle de langage ouvert *sentence-transformers/all-MiniLM-L6-v2* disponible sur la plateforme Hugging Face **[26]**. Celui-ci se spécialise pour des tâches de recherche sémantique ou de clustering et est donc adapté pour la comparaison de phrases. Voici grossièrement la manière dont nous allons l'utiliser :



Nous prenons deux propriétés ou deux mots (ici « King » et « Queen »), pour chacun d'eux, nous les transformons en vecteurs dans un espace sémantique de dimension 384, à l'aide du modèle de langage. Ainsi, le traitement de ces vecteurs peut être fait de la même manière qu'avec ceux issus de plongements lexicaux.

Nous calculons ensuite le cosinus de similarité[8] entre les deux vecteurs, puis nous le comparons à un seuil que nous avons préalablement fixé (suite à plusieurs tentatives, nous avons observé qu'un seuil de 35% est généralement suffisant). Si la valeur de similarité est supérieure au seuil, alors nous les traiterons comme « candidats à la subsomption » et vérifierons leur statut à la main. Si elle est en dessous, nous ne considérerons pas ces couples de propriété.

Cette méthode nous permet de n'évaluer « que » 1538 couples de propriété. La gestion des subsomptions s'est dans un premier temps effectuée à l'aide d'un tableau Excel, dont voici un extrait :

ID R	r1	r2	cos	Relationship status (0 : no relation, 1 : r	Subsomption direction	both_priority
270	participant in	participant	0,957983851	2	opposite	
448	capital	capital of	0,931892184	2	opposite	
1148	use	uses	0,869147301	1	both	from_r1
1487	student of	student	0,867791082	2	opposite	
36	located in the administrative territorial entity	contains administrative territorial entity	0,847486982	2	opposite	
629	owned by	owner of	0,829686711	2	opposite	
1406	candidacy in election	candidate	0,810315324	1	from_r1	
1326	successful candidate	candidate	0,807516861	1	from_r1	
1506	office held by head of government	office held by head of the organization	0,801886511	1	super:office held by head of ...	
1067	replaced by	replaces	0,801548719	2	opposite	
1505	office held by head of government	head of government	0,799414086	1	both	from_r1
1524	head of state	head of government	0,768505857	1	both	from_r1
989	field of work	field of this occupation	0,758549996	1	both	to_r1

Dans celui-ci, nous avons 7 colonnes : l'ID du couple de propriété (qui ne nous intéressera pas par la suite), « r1 » et « r2 » les propriétés du couple, « cos » le cosinus de similarité, « Relationship status » une variable catégorielle disant quelle est la nature de la relation, « Subsomption direction » la direction de la subsomption (unilatérale, bilatérale, etc.) et « both_priority » si la subsomption est bilatérale, mais que l'on ne souhaite vraiment n'en garder qu'un sens, alors lequel ?

L'utilisation de ce tableau permet d'organiser et de catégoriser les couples de propriété de manière « simple » et « exhaustive ». Ainsi, nous observons avec cet extrait presque l'ensemble des types de propriétés prises en charge par celui-ci :

- **« Both » :** les couples de propriété disposant étant régis par une subsomption bilatérale. Nous pouvons par exemple dire que « field of work » peut être subsumé par « field of this occupation », néanmoins si l'on devait vraiment choisir un sens ce serait « to_r1 », c'est-à-dire que la subsomption irait de la deuxième propriété à la première (« from_r1 » dispose du même principe, mais dans l'autre sens).

- **« From_r1 » :** les couples de propriété régis d'une subsomption unilatérale de « r1 » à « r2 ». Par exemple, un « succesful candidate » est un « candidate », mais l'inverse ne tient pas toujours, ainsi un candidat ne sera pas toujours celui qui sera choisi. De la même manière, la relation « To_r1 » existe et correspond à l'autre sens de subsomption.

- **« Opposite » :** les couples de propriété disposant de relations inverses. Ainsi, « participant in » est l'inverse de « participant », un exemple simple de cela est que

[8] Le cosinus de similarité est une mesure de similarité entre deux vecteurs de dimension n très fréquemment utilisé dans ces cadres d'application. Plus d'information sur https://fr.wikipedia.org/wiki/Similarit%C3%A9_cosinus.

l'ONU a pour participant les états membres et que les états membres sont participants à l'ONU. La nature de la relation parcourue (direction) est donc la même, mais un sens différent.

- **« Super : ... » :** ce type de relations indique que les deux propriétés ne sont pas subsumables directement entre-elles, mais qu'elles peuvent être représentées par une autre propriété, parfois non présente dans REBEL, de la même manière que les individus du couple pourraient être perçus comme des instances d'une même classe. Ainsi « Date of Birth » et « Date of Death » pourraient être subsumés par « Date » tout cours.

Après avoir défini le statut de ces propriétés, le tableau est ensuite transformé automatiquement en un texte sous format CSV ou JSON, qui peut ensuite être exporté et importé dans le code Python pour appliquer les propriétés nouvellement créées aux bases de connaissances existantes.

Le code Python permettant de mettre en place ces relations est contenu dans KnowledgeBase.py et est formé à l'intérieur de la classe KB à l'aide d'une nouvelle variable d'instance et de plusieurs autres fonctions.

Celui-ci permet notamment une flexibilité en permettant plusieurs types d'implémentation (je parle ici de la fonction qui applique la subsomption et non celle qui les importe) :

- **« Replace » :** remplace la relation de base par la première subsomption de celle-ci dans la liste de ses subsomptions du fichier JSON.
- **« Weighted » :** remplace la relation de base par la subsomption ayant le cosinus de similarité avec elle le plus élevé.
- **« Add » :** ajoute l'ensemble des subsomptions de la relation sans remplacer la relation préexistante. Il s'agit de la version la plus performante et la plus simple à mettre en place.

Voici un pseudo-algorithme détaillant toutes les étapes de l'implémentation « Add », en choisissant de bien subsumer les relations bilatérales dans les deux sens et en permettant les subsomptions de subsomption à l'infini, de l'importation à l'application des règles :

```
Modules nécessaires dans Python : Json
Fichier du code Python équivalent : KnowledgeBase.py

Méthode de classe KB
Etape I :
Début Algo loadRelationSubsomption (var path (texte), var both_method (texte) ← "both") :
    var SubsomptionJSON (liste(dictionnaire(texte, texte))) ← json.load(path)
    var KB.subsomption (liste) ← []
    Pour chaque subsomption (dictionnaire(texte, texte)) dans « SubsomptionJSON » :
        Si subsomption a pour clef « from_rI » Alors :
            Ajouter à KB.subsomption subsomption
            Ajouter à KB.subsomption subsomption (en inversant la position des membres du couple)
```

```
            Sinon Si subsomption a pour clef « from_r1 » Alors :
                Ajouter à KB.subsomption subsomption
            Sinon Si subsomption a pour clef « to_r1 » Alors :
                Ajouter à KB.subsomption subsomption (en inversant la position des membres du couple)
            Sinon Si la clef de subsomption commence par « super: » Alors :
                Ajouter à KB.subsomption subsomption (en remplaçant r2 par la nouvelle relation)
                Ajouter à KB.subsomption subsomption (en remplaçant r1 par r2 et r2 par la nouvelle relation)
            Sinon Si subsomption a pour clef « opposite » Alors :
                Faire la même chose que pour « both », mais en ajoutant la mention « opposite » dans le triplet
            Fin Si
        Fin Boucle
Fin Algo loadRelationSubsomption

Méthode de classe KB
Etape II :
Début Algo toggleSubsomption (var method (texte) ← "add", var recursiveLimit (entier) ← None) :
    var subRelation (liste(dictionnaire(texte, texte))) ) ← []
    Pour chaque relation (dictionnaire(texte, texte)) dans KB.relation :
        Pour chaque subsomption (dictionnaire(texte, texte)) dans KB.subsomption :
            Si la propriété de relation est égale au sujet de subsomption Alors :
                var cs (liste(dictionnaire(texte, texte))) ← [] #cs : candidateSubsomption
                Si la clef "type" de subsomption est égale à « opposite » Alors :
                    Ajouter à cs relation (en remplaçant type par la tête de subsomption)
                    Ajouter à cs relation (en inversant s et o de relation et en remplaçant p par la queue de subsomption)
                Sinon :
                    Ajouter à cs relation (en rempaçant type par la queue de subsomption)
                Fin Si
                Pour chaque relation dans cs :
                    Ajouter relation à subRelation si relation n'est pas dans subRelation
                Fin Boucle
            Fin Si
        Fin Boucle
        Ajouter subsomption à subRelation si subsomption n'est pas présent dans subRelation
    Fin Boucle
    Quitte la fonction si il n'y a pas de différence entre KB.relation et subRelation # Toute subsomption faite
    KB.relation ← subRelation (byVal)
    Appelle Algo toggleSubsomption récursivement avec les paramètres ("add", None)
Fin Algo toggleSubsomption
```

C'est donc grâce à ce processus (Excel, exportations JSON, importations JSON et application des règles) que nous sommes capables d'étendre les bases de connaissances initiales à la subsomption. Néanmoins, le processus a des limites :

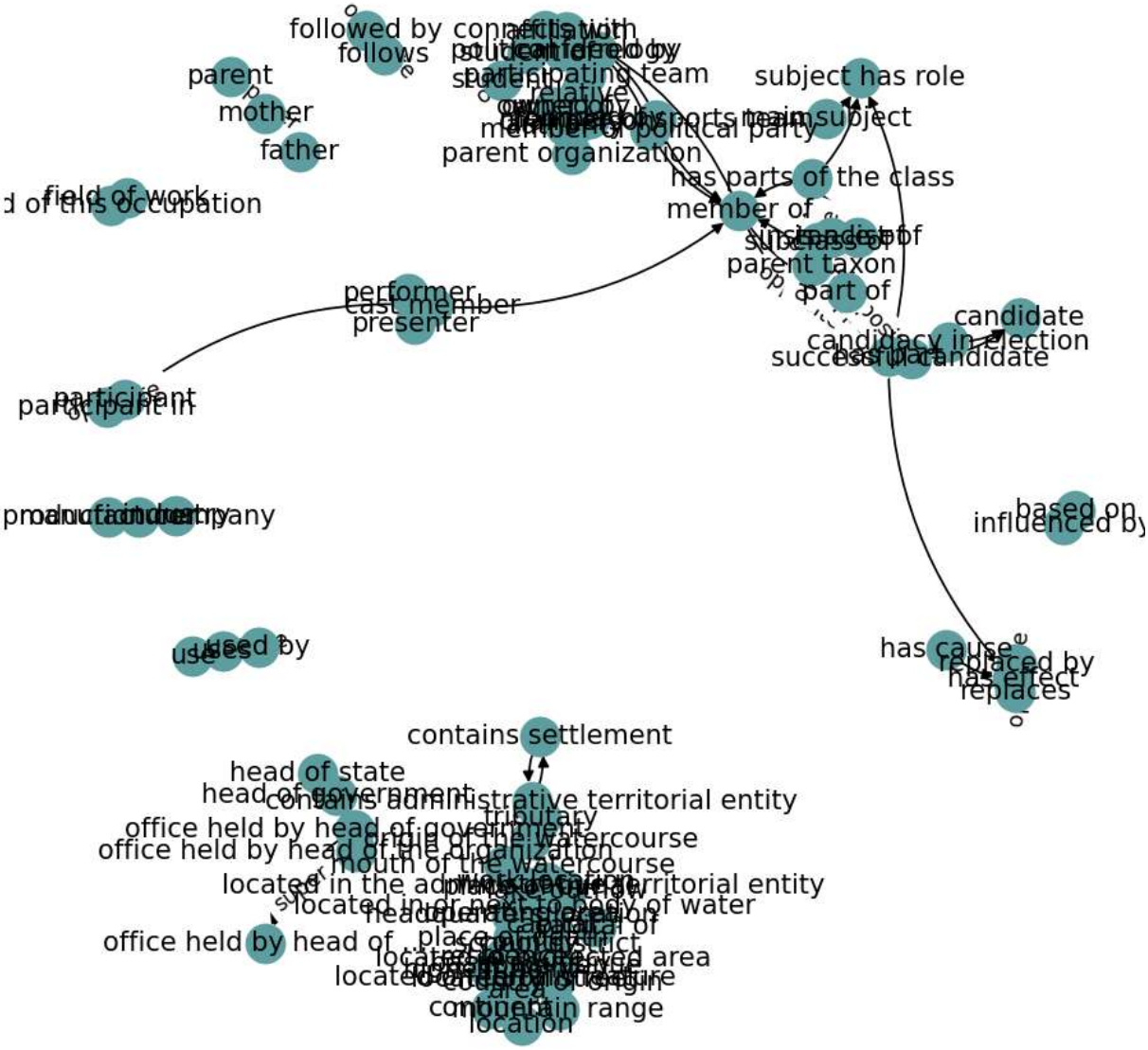
1. Une propriété créée à l'aide de « Super » ne peut pas être référencée pour se faire subsumer (à moins de rajouter une paire entre celle-ci et chaque relation de REBEL, ce qui peut vite devenir contraignant à faire de manière automatique dans Excel). De plus, ces dernières ne peuvent pas être déclarées de manière directe comme transitives.

2. Dans un contexte de recherche, nous souhaiterions surtout avoir quelque chose de flexible et pouvant passer à l'échelle. Or, ce n'est pas le cas ici. En effet, si nous

voulons rajouter des règles plus complexes comme « Le frère de mon père est mon oncle », cela deviendra vite illisible dans le tableur.

Ces problèmes ont été surmontés plus tard et leurs solutions seront détaillées dans la partie sur le web sémantique, RDF, RDFS et OWL.

Il est cependant intéressant de noter que nous pouvons dans ce cadre (certes imparfait), visualiser de manière simple le chemin des subsomptions en regardant le graphe des subsomptions (zoomable sur Python) :



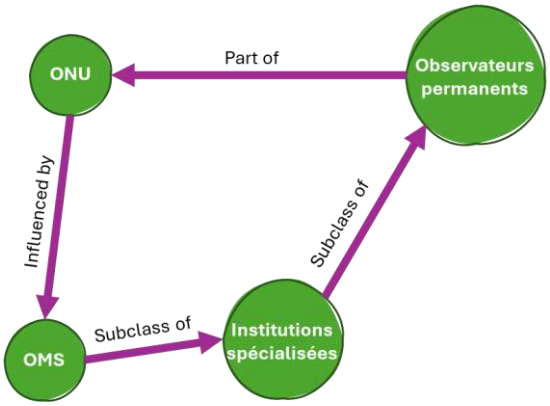
B.3.c – Création de la base de connaissances, relations de transitivité

Avant de décrire l'implémentation étant derrière la transitivité, nous avions défini une propriété transitive dans la partie **[I – B]** ainsi :

$$(< s_0; p_0; o_0 > \Rightarrow < o_0; p_1; o_1 >) \Rightarrow < s_0; p_{0V1}; o_1 >$$

Soit, qu'une relation est transitive si et seulement si nous pouvons en tirer une implication entre plusieurs triplets à la chaîne (c'est-à-dire que l'objet du précédent constitue la même entité que le sujet du triplet courant).

Néanmoins, comme dit plus tôt il ne s'agit **que** d'une implication. Cela ne peut pas réellement constituer une relation de transitivité. En effet, le problème principal avec cette définition se situe sur le point que les propriétés des deux triplets n'ont pas la nécessité d'être les mêmes. Cela peut poser plusieurs problèmes, voici un exemple :



Ici, nous pouvons définir toutes les propriétés comme ayant une capacité d'impliquer quelque chose, une sous-classe d'une sous-classe reste une sous-classe, se faire influencer par quelque chose qui est lui-même influencé par une autre chose revient à se faire influencer par autre chose et si A fait partie de B et que B fait partie de C, alors A fait partie de C.

Sur le principe, cela peut se défendre, mais si l'on applique notre définition, nous aurions un lien à tracer entre l'ONU et les institutions spécialisées telles que l'ONU est influencée par les institutions spécialisées (crédible), mais nous pourrions aussi tracer le lien que l'ONU est une sous-classe des institutions spécialisées, ce qui est faux.

D'autres problèmes peuvent apparaître, entre autres : les liens réflexifs **[27]** tels que : « l'OMS est une sous classe de l'OMS ». Cela pose problème pour de multiples raisons, mais principalement pour la baisse de performance en temps de calcul pour l'évaluation du modèle et peut impliquer des baisses de performance pour l'estimation des connaissances de l'utilisateur en raison de la formule utilisée plus tard (voir partie sur l'évaluation des modèles et **Erratum**).

Il y a également la question de « comment choisit-on la propriété à conserver ? », la machine ne sait pas à l'avance si un triplet est « vrai » ou « faux ».

Note : D'un point de vue plus technique, nous pouvons également soulever à partir de cet exemple des problématiques liées aux subsomptions. En effet, si l'on n'autorise les transitivités qu'entre les triplets disposant des mêmes propriétés, nous serions tentés de subsumer « Part Of » en « Subclass Of » pour avoir un lien tel que les institutions spécialisées sont une sous-classe ou une partie de l'ONU. Or, cela n'est pas possible de manière rigoureuse, car le lien « sous-classe de » se situe entre des classes, mais le lien « partie de » se situe entre des instances. Ainsi, l'ONU n'est pas une classe, mais une instance de la classe « Organisation Internationale ». Nous ne détaillerons pas cette subtilité dans le rapport, mais nous garderons cela dans un coin de la tête lorsque nous retravaillerons sur les subsomptions (partie web sémantique, RDF, RDFS, OWL).

Pour éviter ces problèmes, nous allons donc dissocier la définition de l'implication ci-dessus et celle de la transitivité **[28]** que nous définirons par la suite ainsi :

$$(< s_0; p; o_0 > \Rightarrow < o_0; p; o_1 >) \Rightarrow < s_0; p; o_1 >$$

Soit, qu'une relation est transitive si et seulement si nous pouvons en tirer une implication entre plusieurs triplets à la chaîne (c'est-à-dire que l'objet du précédent constitue la même entité que le sujet du triplet courant) **ET** que la propriété de TOUS les triplets constituant la chaîne est le même.

Maintenant que nous avons défini cela, il nous faut l'implémenter. À ces fins, j'ai repris le fichier contenant les propriétés disponibles dans REBEL, puis les ai transférées dans un tableau Excel dans lequel j'ai indiqué pour chacune des propriétés à la question « Est transitive ? » oui ou non (ici, il n'y a pas de couple, donc il n'y a que 230 propriétés à vérifier).

Comme pour la subsomption, les données sont ensuite transformées dans un fichier JSON dont voici un extrait :


```
[
   {"relation": "country", "transitivity": "0"},
   {"relation": "located in the administrative territorial entity", "transitivity": "1"},
   {"relation": "contains administrative territorial entity", "transitivity": "1"},
   {"relation": "date of birth", "transitivity": "0"},
   {"relation": "sport", "transitivity": "0"},
   {"relation": "instance of", "transitivity": "1"},
   {"relation": "point in time", "transitivity": "0"},
   {"relation": "date of death", "transitivity": "0"}, …
]
```


Nous avons finalement une liste avec pour chaque élément : le nom de la relation et si la relation est transitive ("0" désigne les relations non transitives et "1" les relations transitives). La liste pourra par la suite être importée dans Python par un processus similaire à l'import des subsomptions.

Note : ici nous aurions pu coder "0" et "1" en entier ou en booléen, mais par consistance, nous les avons gardés en texte et les convertiront en booléen dans Python directement.

Comme pour la subsomption, il existe plusieurs modes d'implémentation de la transitivité, nous pouvons régler la limite de longueur de la chaîne de récursivité (par défaut nous la laissons sans limites), définir si nous autorisons les relations réflexives (non par défaut) et la méthode : **« first »** et **« last »** qui définissent quelle propriété est prise dans le cadre d'une implication, enfin, « **unique** » applique la vraie définition de la transitivité.

Il existe également deux manières équivalentes d'appliquer la transitivité ou l'implication : par récursivité (implémentée pour les implications) ou de manière itérative (implémentée pour les transitivités et les implications).

Dans un premier temps, nous avons mis en place la méthode récursive, nous allons donc la décrire en première et expliquer pourquoi nous sommes passés sur une version itérative après. Dans les deux cas, nous importons les transitivités de la même manière, similairement à la subsomption et les stockons dans la variable d'instance KB.transitivity.

Voici un pseudo-algorithme simplifié de la méthode récursive sans limitations dans la longueur de la chaîne et en interdisant les relations réflexives :

```
Fichier du code Python équivalent : KnowledgeBase.py

Méthode de classe KB
Début Algo __toggleTransitivity_recursive(var listeTransitivité (liste(dictionnaire))) → liste(dictionnaire) :
    listeTransitivitéCopy (liste) ← listeTransitivité (byVal)
    Pour chaque relations « r » dans « listeTransitivité » :
        # Récupère les triplets ayant une propriété transitive et pour sujet l'objet de « r »
        var i (liste) ← KB.findOutgoingRelationFromHead(queue de « r », True)
        Ajouter à « listeTransitivité » tous les élements de « i » n'étant pas dans « listeTransitivité »
    Fin Boucle
    Si listeTransitivité identique à listeTransitivitéCopy Alors : #On a trouvé toutes les transitivités
        Retourner listeTransitivité
    Fin Si
    Appelle Fonction __toggleTransitivity_recursive avec le paramètre (listeTransitivité)
Fin Fonction __toggleTransitivity_recursive

Début Algo toggleTransitivity(var  method (texte) ← "first") :
    var pointFait (liste(texte)) ← []
    var listeTransitivité (liste(liste(dictionnnaire))) ← []
    Pour chaque relations « r » dans « KB.transitivity » :
        Ajouter « r » dans « pointFait » si « r » n'est pas dans « pointFait » sinon itération suivante
        var t (liste) ← KB.findOutgoingRelationFromHead(queue de « r », True)
        Ajouter __toggleTransitivity_recursive(« t ») à « listeTransitivité » si « t » n'est pas vide
    Fin Boucle
    Pour chaque « cheminTransitif » dans « listeTransitivité » :
        Pour chaque « relation » dans « cheminTransitif » :
            Ajouter à KB.relation le triplet avec en "head" la tête du triplet de départ du cheminTransitif,
                                                 en "type" le type du triplet de départ du cheminTransitif
                                                 et en "tail" la queue de « relation »,
                                                 si la relation n'existe pas dans KB.relation
        Fin Boucle
    Fin Boucle
Fin Algo toggleTransitivity
```

Avec ce type d'algorithme, nous sommes donc capables de mettre en place la transitivité dans la base de connaissances. Néanmoins, l'algorithme récursif présenté ci-dessus regorge de plusieurs problématiques, intrinsèques ou non à la récursivité. Les plus importantes sont les suivantes :

1. Le pseudo-algorithme présenté **ne met pas en place la transitivité** comme nous l'avons défini, mais l'implication. En effet, à ce moment-là du stage, la nuance n'avait pas encore été soulevée.

2. La récursivité, bien qu'adaptée dans ce cadre, présente en Python des problèmes quant à **l'optimisation de la mémoire et de la vitesse de calcul**. Cela réduit ainsi les possibilités de mise à l'échelle pour les documents de taille importante, en augmentant le temps de calcul de manière considérable. Lors de l'évaluation du modèle, nous nous sommes aussi rendus compte d'un blocage pour certains individus, pour lesquels l'évaluation était simplement impossible en appliquant cet algorithme pour l'implication (même après huit heures pour certains individus, nous n'obtenions pas de résultat).

3. En Python, même si anecdotique dans la plupart des cas, la profondeur de la récursivité est limitée à 1000 (dépend aussi des versions). Celle-ci peut être augmentée au besoin, mais nous limiterions une sécurité native de Python. Ainsi, nous ne pouvons pas avoir une chaîne de transitivité d'une profondeur dépassant cette limite.

Pour résoudre ces problèmes en profondeur, nous devons trouver une autre manière de faire que la méthode récursive.

Pour cela, nous pouvons nous baser sur un questionnement posé sur StackOverflow et sur la conjecture de Church-Turing **[29, 30]**, qui nous disent *très grossièrement* que toutes fonctions récursives peuvent être transformées en fonctions itératives. Ces fonctions itératives ont l'avantage, en Python, d'être bien mieux gérées d'un point de vue « optimisation de mémoire » et « temps de calcul ». Nous avons donc décidé de mettre en place cet algorithme itératif en faisant en sorte de pouvoir aussi calculer la transitivité avec la bonne définition cette fois-ci.

Note : Pour mettre en place l'itérativité ici, nous gérons **manuellement** une liste où le dernier élément qui rentre est le premier qui sort, c'est ce qu'on appelle un « stack ». Dans le cadre de la récursion, celui-ci existe aussi, mais est géré de manière « automatique » sous une autre forme. Voir https://stackoverflow.com/questions/159590/way-to-go-from-recursion-to-iteration, pour en savoir plus sur la méthode utilisée.

Voici un dernier pseudo-algorithme simplifié de la méthode itérative pour une transitivité, sans limites de longueur de chaîne et en interdisant les relations réflexives :

```
Fichier du code Python équivalent : KnowledgeBase.py

Méthode de classe KB
Début Algo toggleItteративelyTransitivity(var  method (texte) ← "unique") :
    var pointFait (liste(texte)) ← []
    var listeTransitivité (liste(liste(dictionnaire))) ← []
    var stack ← []
    Pour chaque relation « r » dans « KB.relation » :
        Ajouter « r » dans « pointFait » si « r » n'est pas dans « pointFait » sinon itération suivante
        Ajouter KB.findOutgoingRelationFromHead(queue de « r », True) à « stack »
        Tant que stack n'est pas vide :
            var obj (liste) récupère et retire la relation en haut de la pile « stack »
            var baseObj (liste) ← obj (byVal)
            Pour chaque relation « i » dans une copie de obj :
                var outTransitivity ← KB.findOutgoingRelationFromHead(queue de « r », True)
                Pour chaque relation « j » dans outTransitivity :
                    Ajouter « j » à obj si le prédicat de « i » et de « j » sont les mêmes
                Fin Boucle
            Fin Boucle
        Ajoute « obj » à « listeTransitivité » et sort de la boucle si obj est égale à baseObj
        Ajoute « obj » en haut de la pile « stack »
    Fin Boucle
    Pour chaque « cheminTransitif » dans « listeTransitivité » :
        Pour chaque « relation » dans « cheminTransitif » :
            Ajouter à KB.relation le triplet avec en "head" la tête du triplet de départ du cheminTransitif,
                                                en "type" le type du triplet de départ du cheminTransitif
                                                et en "tail" la queue de « relation »,
                                                si la relation n'existe pas dans KB.relation
        Fin Boucle
    Fin Boucle
Fin Algo toggleItteративelyTransitivity
```

C'est donc à travers toutes ces étapes (Excel, JSON et itérativité) que nous sommes capables d'étendre les bases de connaissances de $RULK_{KG}$ à la transitivité. Néanmoins, comme pour la subsomption, ce processus n'est pas « propre », dans le sens où il n'obéit pas aux standards existants, rendant encore une fois le passage à l'échelle compliquée.

Avant de rentrer en détail sur cette standardisation, nous devons d'abord vérifier si ce qui a été effectué contribue à une amélioration de la connaissance ou non.

C – Expériences et résultats

Avant d'expliquer le processus d'évaluation du modèle, il faut d'abord comprendre le jeu de données va nous permettre cette évaluation.

C.1 – Présentation du jeu de données

Le jeu de données a été obtenu depuis une étude se concentrant sur les sessions de recherche impliquant 127 « Crowd-worker » ayant pour mission de rechercher des informations sur 7 sujets de différents domaines. Le moteur de recherche utilisé pour cette étude a été construit au-dessus de SeachX **[31]**, un Framework pour la recherche interactive d'information. Le jeu de données inclue aussi certaines données sur leurs

comportements, telles que la longueur de la requête, le nombre de requêtes, le nombre de clics et la durée de session.

De plus, nous retrouvons également ce que l'on appellera au long du rapport de stage le « gain de connaissance réel » de l'utilisateur. Évalué à l'aide de deux tests de connaissance sur le sujet confié, un avant les recherches et l'autre après, la différence entre ces résultats nous donne une mesure du gain de connaissance nommé : « **Absolute Learning Gain** » (**ALG**) représentant le montant total des connaissances acquises, ainsi qu'une autre mesure pouvant en être extraite : « **Realized Potential Learning** » (**RPL**), soit une valeur entre 0 et 1 quantifiant le pourcentage de connaissances acquises par l'utilisateur par rapport à son « **Maximum Learning Gain** » (**MLG**), soit le montant de connaissances maximales que l'utilisateur peut gagner par rapport à ses connaissances cibles et ses connaissances de base sur une session d'apprentissage.

Note : Ce jeu de données est le même que celui utilisé dans les dernières versions de RULK, cela nous permettra de faire les comparaisons par rapport aux anciens modèles.

En outre, nous utiliserons pour l'évaluation du modèle, les graphes précédemment extraits pour $RULK_{KG}$ **[04]**, auquel nous inférerons les connaissances supplémentaires. En effet, réextraire les graphes prend beaucoup trop de temps (estimation minimum à 9 jours 24h/24h).

Parenthèse : C'est aussi pour cela que nous n'évaluerons pas le stemming fait plus tôt qui nécessiterais de regénérer les graphes[9], cela constitue donc une piste d'amélioration pour la fusion des entités, néanmoins, il faudra faire attention pour celle-ci, car renvoyer les mots à leur racine peut flouter leurs sens.

C.2 – Méthode et protocole d'évaluation

Dans le cadre de ce stage, le gros du système d'évaluation existe déjà. Il n'a donc fallu que quelques modifications mineures pour implémenter le système de subsomption et de transitivité.

Voici donc les étapes principales du code permettant l'évaluation du modèle (pré-prise en compte des connaissances implicites) :

- **Étape 1 :** Nous importons les données,
- **Étape 2 :** Récupération des triplets et ajout aux connaissances de l'utilisateur, pour chaque document visité de chaque utilisateur (il s'agit donc de σ).
- **Étape 3 :** Récupération des triplets de l'article Wikipédia associé.
- **Étape 4 :** Calcul de la similarité entre les triplets de l'article et ceux de l'utilisateur.

[9] Dans les faits, nous aurions pu le faire à la fin du stage, mais d'autres pistes d'exploration semblaient plus intéressantes pour augmenter les performances des modèles.

Comme dit plus tôt lors de la présentation des instances de RULK, $RULK_{KG}$ n'utilise pas le cosinus de similarité pour mesurer la ressemblance entre les connaissances de l'utilisateur et les connaissances cibles. En effet, cette mesure ne correspond pas aux graphes. Une nouvelle mesure avait donc été inventée **[04]** : $TripletSim(G_t, G_c) = \frac{N_{identical(G_t,G_c)} + N_{semantic\_trp(G_t,G_c)}}{N_{G_t}}$,où $N_{identical(G_t,G_c)}$ représente le nombre de triplets identiques entre les deux graphes, $N_{semantic\_trp(G_t, G_c)}$ le nombre de triplets « similaires » entre les deux graphes, où la notion de « similaire » se traduit par le calcul du cosinus de similarité entre les versions textuelles des triplets (<Arno, étudie à, Université Côte d'Azur> → « Arno étudie à Université Côte d'Azur ») avec l'aide du modèle de langage `en_core_web_md` de SpaCy **[32]**. Le triplet est ensuite considéré comme similaire si la mesure dépasse un seuil fixé (80% pour pouvoir comparer avec les anciennes versions).

- **Étape 5 :** Calcul de la corrélation.

  Nous calculons la corrélation entre l'ALG ou le RPL (soit les connaissances « réelles » de l'utilisateur) à celles estimées par le graphe. La mesure de corrélation choisie lors des précédentes études et auquel nous resterons attachés pour des soucis de comparabilité est la corrélation de Pearson **[61]**.

- **Étape 6 :** Combinaison des modèles.

  Finalement, les instances de RULK peuvent être combinées afin d'avoir un modèle reflétant plusieurs aspects des connaissances de l'utilisateur. Cela se fait en optimisant le poids de chaque modèle en maximisant l'ALG ou le RPL en utilisant la formule : $\sum_i w_i RULK_i \ (s.c. \sum_i w_i = 1)$.

Pour ajouter les inférences aux graphes, il suffit donc d'instancier au sein de **l'étape 2** ou juste avant **l'étape 3**[10] un objet de classe **KB**, de lui affecter comme relations le graphe, de faire les inférences avec l'objet de classe KB et de récupérer les relations. Cela ne rajoute donc effectivement que 14 lignes de code sur les 191 lignes initialement présentes, ce qui rend l'ajout comparativement léger en représentant 7,11% du total du code de l'évaluation.

Remarque : Même si une des missions du stage consiste en l'intégration de la nouvelle base de connaissances dans le cadre existant, celle-ci n'a pas été nécessaire dans la mesure où la semaine initiale pour la détection des problématiques, m'a permis de démarrer du bon pied, en effectuant les modifications directement dans le cadre existant.

[10] Attention, les résultats sont différents en fonction de l'endroit de l'implémentation, ainsi, dans l'étape 2, les inférences seront faites à chaque nouveau document et au sein de ces derniers seulement. Tandis qu'avant l'étape 3, les inférences se font aussi entre les graphes, mais à la fin uniquement.

Au niveau du protocole en tant que tel, nous lançons plusieurs expérimentations (avec/sans subsomption/transitivité, sur les connaissances de l'utilisateur et/ou sur les connaissances cibles). Nous enregistrons ensuite chaque résultat dans un document CSV contenant le RPL ou l'ALG ainsi que les connaissances estimées avec un titre contenant le seuil auquel a été fait l'expérience, la portée des inférences et le type d'inférence. Puis, nous classons les résultats de l'expérimentation en fonction de leur corrélation avec le RPL, leur corrélation avec l'ALG et le temps de calcul. Éventuellement, nous combinons le modèle engendré avec les autres instances de RULK. Enfin, nous comparons ces modèles aux versions antérieures de RULK.

Aussi, pour être sûr de la fiabilité des résultats, nous effectuerons une reproduction de l'étude sur laquelle nous nous basons **[04]**. Ainsi, si les résultats de la reproduction sont différents, nous saurons que les résultats que l'on tirerait du nouveau modèle seraient justifiés par autre chose que les modifications apportées.

Note : Pour chaque corrélation calculée, un test de nullité de la corrélation est effectué. Cependant, par oublis, les p-values des tests de nullité des corrélations n'ont pas été reportées dans le tableau Excel, mais ont toutes été vérifiées et se situent toutes en dessous de 5% et pour la plupart en dessous de 1%.

C.3 – Résultats

Pour commencer, sachez que l'ensemble des résultats sont disponibles dans le classeur Excel « Résultats améliorations.xlsx » **[34]**. Nous ne montrerons ici que les plus importants.

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KW)	0,3022	0,3086	N/A
RULK (LM)	0,2955	0,2923	N/A
RULK (NE)	0,0931	0,1180	N/A
RULK (KG, référence)	0,2651	0,2362	N/A
RULK (KG, reproduction)	0,2194	0,1879	30 minutes

Ici, nous observons que la reproduction des résultats ne donne pas ceux escomptés. Nous pouvons conclure qu'il y a donc un changement entre les deux versions, ces derniers peuvent être de trois origines principales :

- Mauvaises données de départ,
- Erreur dans le code,
- Versions de Python/packages différentes.

Après vérification, il ne s'agissait pas des données de départ, qui étaient bien les mêmes que celles de l'étude. Il ne s'agissait pas non plus d'erreurs dans le code.

Finalement, il s'agissait de versions différentes, voici plus précisément les différences :

Étude	Reproduction
Python 3.9.13	Python 3.12.0
SpaCy 3.5.2	SpaCy 3.7.4
en_web_md 3.3.0	***en_web_md 3.7.1***
NumPy 1.26.3	NumPy 1.26.4

La solution est donc bien simple, il suffit de changer les versions des modules. Or, les résultats de la reproduction restent des résultats et soulèvent la dépendance du cadre RULK par rapport au modèle de langage utilisé dans la mesure de similarité des graphes.

Il faut donc trouver un moyen de garder les deux versions et de choisir laquelle exécuter quand nous le souhaitons. Malheureusement, PIP **[69]**, le gestionnaire de package de Python ne permet pas une telle chose. La solution aura donc été la mise en place d'un environnement virtuel, sur lequel nous avons installé tous les packages à la bonne version. Vérifions avec la nouvelle reproduction :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KW)	0,3022	0,3086	N/A
RULK (LM)	0,2955	0,2923	N/A
RULK (NE)	0,0931	0,1180	N/A
RULK (KG, référence)	0,2651	0,2362	N/A
RULK (KG, reproduction)	0,2651	0,2363	37 minutes

Ici, les résultats de la reproduction sont maintenant identiques à ceux de l'étude. Il est tout de même intéressant de soulever que le temps d'exécution a légèrement augmenté (note : c'est aussi le cas en général pour tous les tests effectués par rapport au mauvais environnement virtuel).

Regardons maintenant l'ensemble des résultats :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, reproduction)	0,2651	0,2363	37 minutes
Subsomptions & Autres utilisateur et cible	0,2050	0,1709	15 heures, 40 minutes
Subsomptions & Autres utilisateur	0,2197	0,1987	2 heures, 50 minutes
Transitivités utilisateur et cible	0,2652	0,2388	40 minutes
Transitivités utilisateur	0,2575	0,2293	38 minutes
Implications utilisateur et cible	0,2482	0,2188	47 minutes
Implications utilisateur	0,2552	0,2268	44 minutes
Subsomptions & Autres, Transitivités utilisateur et cible	N/A	N/A	N/A

Avec ce tableau, nous observons plusieurs choses :

1. La subsomption & autres seule ne permet pas d'obtenir une amélioration dans la corrélation en ALG ou en RPL, pire, nous obtenons une baisse moyenne 19,90% dans la corrélation en ALG et de 21,79% en RPL quand ce type de relations est impliqué.

2. La transitivité ne semble pas apporter d'amélioration significative en ALG ou en RPL par rapport à la référence quand nous l'appliquons au graphe de l'utilisateur et à celle des connaissances cibles. De plus, celle-ci offre une baisse 2,87% en ALG et de 2,96% en RPL, par rapport à la référence quand elle n'est impliquée que dans le graphe de l'utilisateur.

3. L'implication semble effectivement moins performante que la transitivité simple avec une baisse moyenne par rapport à cette dernière de 3,69% en ALG et de 4,81% en RPL.

4. Le temps d'exécution pour appliquer la subsomption & autres sur le graphe de l'utilisateur et le graphe de la cible est très conséquent. Cela empêche notamment l'évaluation quand la transitivité et la subsomption & autres sont à implémenter en même temps.

La question est maintenant de comprendre pourquoi les résultats sont si mauvais et pourquoi cela prend beaucoup de temps à exécuter.

Voici deux **hypothèses** qui pourraient expliquer ces mauvais résultats :

- Le modèle de langage, bien qu'adapté pour la précédente version de $RULK_{KG}$, ne l'est plus pour cette version disposant d'inférences.
- Le paramétrage des subsomptions n'est pas bon.

La première serait une bonne chose dans la mesure où cela voudrait dire que j'ai bien effectué mon travail. Néanmoins, même si c'était le cas, cela poserait des problèmes d'interprétabilité dans la comparaison des futurs résultats. De plus, la plupart des modèles de langage se basant sur des méthodes telles que le Deep Learning, il est compliqué, voire impossible de savoir ce qui aurait pu causer ces différences.

Note : Après test du modèle *transformers/all-MiniLM-L6-v2*, utilisé précédemment dans le cadre de la présélection des subsomptions, des résultats similaires (toujours mauvais) sont présents, ce qui nous conforte dans l'idée de la seconde hypothèse.

Ainsi, nous allons travailler sur la deuxième hypothèse qui semble gérable « plus simplement ». Néanmoins, après avoir commencé à retravailler sur les subsomptions, je me suis vite rendu compte que la non-standardisation du système créé pour classifier les subsomptions allait poser problème, que ce soit pour l'extensibilité future ou la lisibilité.

Il faut donc revoir **l'ensemble** du système de subsomption et de transitivité.

D – Web Sémantique : RDF, RDFS, OWL et SPARQL

Au cours de ce stage et au fur et à mesure des réunions hebdomadaires, les termes RDF, SPARQL et OWL revenaient de temps à autres lorsque je parlais de subsomptions et de transitivités. Cela permettrait entre autres de compléter des graphes de manière plus simple et standardisée, mais nécessitait du temps pour apprendre. Nous l'avions donc initialement relégué en « Bonus ».

Néanmoins, comme expliquée précédemment, le besoin de lisibilité et de standardisation se faisant sentir, j'ai demandé à M. CARTON, étudiant en 3ème année de BUT SD, les ressources qu'il avait utilisées pour apprendre SPARQL dans le cadre de son alternance au laboratoire.

Il m'a alors renvoyé un lien vers un cours de l'INRIA **[35]** (disponible à tous sur France Université Numérique) sur le web sémantique **[36]**, que j'ai suivi pendant une semaine avec l'accord de mes maîtres de stage.

Dans les parties suivantes, je vais donc, en me basant sur ce cours, retranscrire ce qu'est le web sémantique et comment cela pourrait nous aider dans le cadre de nos missions.

D.1 – Présentation du web sémantique

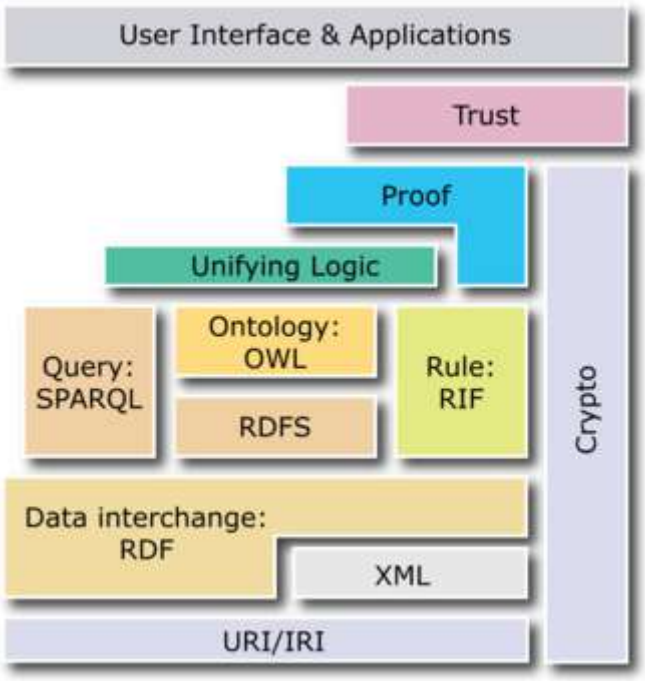
Avant de parler de web sémantique, revenons rapidement à la base du web tel que nous le connaissons aujourd'hui. Celui-ci repose sur trois composantes principales :

- **L'identification et l'adressage :** généralement l'URL,
- **Le protocole de communication :** généralement HTTP,
- **Un langage de représentation :** généralement HTML.

Si l'on combine HTML et l'URL, nous obtenons une référence, l'URL et HTTP une adresse, HTTP et HTML une communication, enfin si l'on combine les trois, nous obtenons le WEB.

À partir de là, nous pouvons faire émerger le concept de données liées, c'est-à-dire une ressource qui pointe vers une autre ressource, qui pointe vers une autre ressource, etc. Ces données ont l'avantage comme leur nom l'indique de lier les données entre elles, à l'instar des hyperliens sur les pages HTML. Ce sont celles-ci qui sont à la base du « web sémantique ».

Il reste néanmoins des différences sur ses principales composantes par rapport au « web classique ». En effet, celui-ci dispose de ses propres langages de représentation, mais aussi d'un autre moyen d'identification. Voici une pile, trouvée sur Internet, permettant de représenter le web sémantique de manière simplifiée :



Celui-ci se base donc sur les **U**niform **R**esource **I**dentifier (URI) et les **I**nternationalized **R**esource **I**dentifier (IRI) **[37]**, qui sont pour faire simple des URL avec pour exception qu'ils ne sont pas dédiés à identifier des choses uniquement présentes sur le web, mais aussi celles de la réalité. Ainsi, l'être humain Arno LESAGE n'existe pas en tant que tel sur le web, tout comme le laboratoire I3S, néanmoins, ces deux-là peuvent être identifiés sur le web via un URI ou un IRI.

Ensuite vient E**x**tensible **M**arkup **L**anguage (XML) **[38]** et **R**esource **D**escription **F**ramework (RDF) **[39]** qui constituent des langages de représentation de données, nous n'allons ici pas vraiment nous intéresser au premier, mais uniquement au second.

D.1.a – Présentation du web sémantique, RDF

RDF est donc un langage de représentation des données qui permet de les présenter sous forme de graphes (étant donné qu'elles sont liées). Il a pour but de décrire de manière simple des ressources et leurs métadonnées, afin de permettre le traitement automatique de ces descriptions. Il existe plusieurs standards du W3C **[40]** qui régissent ce langage de représentation (RDF/XML, Turtle, N-Triples, etc. **[41, 42, 43]**), mais ils gardent globalement la même « racine » au niveau de leur syntaxe : des triplets $< subject, predicate, object >$ où le sujet constitue la ressource à décrire, le prédicat la propriété applicable à la ressource et l'objet qui peut être soit une valeur (Literal) ou une autre ressource.

Voici donc un même graphe disant qu'Arno est une personne sous différentes syntaxes :

RDF/XML

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <rdf:Description rdf:about="http://example.org/Arno">
    <rdf:type rdf:resource="http://example.org/Person"/>
  </rdf:Description>

</rdf:RDF>
```

Turtle

```
@base <http://example.org/> .
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<Arno> rdf:type <Person> .
```

N-Triples

```
<http://example.org/Arno> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://example.org/Person> .
```

Avec cela, nous observons bien en vert la description du sujet sous sa forme d'URI avec RDF/XML et N-Triples (http://example.org/Arno[11]). Pour Turtle, nous avons un simple <Arno>. Ceci est dû au fait que nous avons défini un nom de domaine en **base**, ainsi la première notation est équivalente à la deuxième en Turtle.

Ensuite, nous identifions bien en jaune la propriété qui lie les ressources (note : la propriété `rdf:type` peut être remplacée par « a » dans Turtle).

Enfin, en orange, nous retrouvons l'objet, soit le fait qu'Arno est de type personne.

Dans ce type de syntaxe, il faut bien comprendre que nous ne sommes pas limités par le nombre de triplet pour un même sujet. De plus, nous n'effleurons ici qu'a peine les possibilités que nous développerons par la suite. En effet, RDF en lui-même ne permet pas de faire des inférences, mais juste de décrire des ressources. Il faut donc encore monter d'un niveau dans la pile.

D.1.b – Présentation du web sémantique, RDFS, OWL et SPARQL

Dans ce nouveau niveau de la pile, nous avons affaire à deux parties distinctes : le raisonnement avec **RDFS**chema (RDFS) et **W**eb **O**ntology **L**anguage (OWL), ainsi que le requêtage avec **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage (SPARQL) **[72, 73, 74]**.

Dans la suite du stage, nous n'utiliserons pas de manière récurrente SPARQL, mais il me semble intéressant de mentionner a minima comment ce langage de requête fonctionne. Commençons donc par celui-ci.

[11] Le lien ne marche pas, car je n'ai pas publié ce graphe sur le web à cette adresse. En effet, nous n'avons pas pour objectif de publier un tel graphe, mais juste de s'en servir pour faire des inférences **localement**.

SPARQL est construit pour pouvoir interroger des graphes RDF (peu importe le format), cela peut avoir des applications assez intéressantes dans la mesure où, plutôt que d'interroger une base de données relationnelle, nous interrogeons un graphe, ce qui peut soulever d'autres types de données. Voici un exemple d'utilisation de SPARQL sur un graphe RDF (TURTLE) :

Graphe (RDF – Turtle)

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<Arno> a <Person> ;
    foaf:mbox "arno.lesage1@gmail.com";
    foaf:familyName "Lesage" ;
    foaf:knows <Hadi> .

<Hadi> a <Person> ;
    foaf:knows <Arno> ;
    foaf:knows <Unknown> .

<Unknown> a <Person> ;
    foaf:knows <Arno> ;
    foaf:knows <Célia> .

<Célia> a <Person> .
```

Requête (SPARQL)

```
BASE <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?s
WHERE {
    {
        ?s foaf:knows ?p .
        ?p foaf:knows ?s .
    } UNION {
        ?s foaf:knows <Célia> .
    }
}
```

Résultats

```
<Hadi>
<Arno>
<Unknown>
```

Le graphe RDF décrit quatre personnes : « Arno », « Célia », « Hadi » et « Unknown » en utilisant FOAF **[44]**, un schéma RDFS/OWL permettant de décrire des individus. Ainsi, nous voulons savoir qui se connaît mutuellement (Arno connaît Hadi qui connaît Arno) ou qui connaît Célia

Note : Toutes ressemblances à la réalité dans le schéma serait **fortuite**.

Pour cela, nous écrivons une requête SPARQL qui récupère l'identifiant des ressources des personnes qui recoupe l'une des conditions que nous avons fixées. Au niveau de la syntaxe du langage, nous pouvons définir une base et un préfixe, comme dans la syntaxe Turtle de RDF. Nous désignons également les variables en les précédents d'un point d'interrogation. Ces dernières permettent de désigner des triplets qui matchent les patterns que l'on souhaite. Enfin, par défauts, tous les triplets que l'on cherche se cumulent avec un « **ET** » logique et se terminent par un point.

De plus, il est intéressant de noter qu'au-delà de ces commandes simples, SPARQL permet d'effectuer des vérifications sur les Littéraux, de chercher pour des chaînes de triplet particulières, de chercher dans plusieurs graphes en même temps, de mettre un jour un graphe, supprimer un graphe, voire en créer. En somme, il s'agit du langage SQL pour les données sous forme de graphe.

Bien que pouvant être assez utile, notamment pour ses capacités à vérifier des chaînes particulières ou mettre à jour un graphe, ce qui nous intéresse pour notre mission de redéfinir les subsomptions, les transitivités, etc., il existe cependant les outils de raisonnement que sont RDFS et OWL qui sont plus pratiques dans notre cas.

Le premier, RDFS, est comme son nom l'indique un schéma RDF. Plus clairement, cela veut dire que l'on peut définir dans celui-ci des propriétés pour ses propres classes ou prédicats. Ici, nous ne nous intéressons pas à la classe des entités dans la mesure où cela nécessiterait de la déterminer depuis le graphe, ce qui n'est pas aisé. Nous ne nous servirons donc de RDFS que pour définir des propriétés sur les relations.

De RDFS, nous n'utiliserons que la commande rdfs:subPropertyOf, qui se traduit en logique du premier ordre ainsi : $x\mathcal{R}_1 y \Rightarrow x\mathcal{R}_2 y$. Cela rend cette propriété équivalente à ce que nous appelons depuis tout à l'heure la « subsomption unilatérale ».

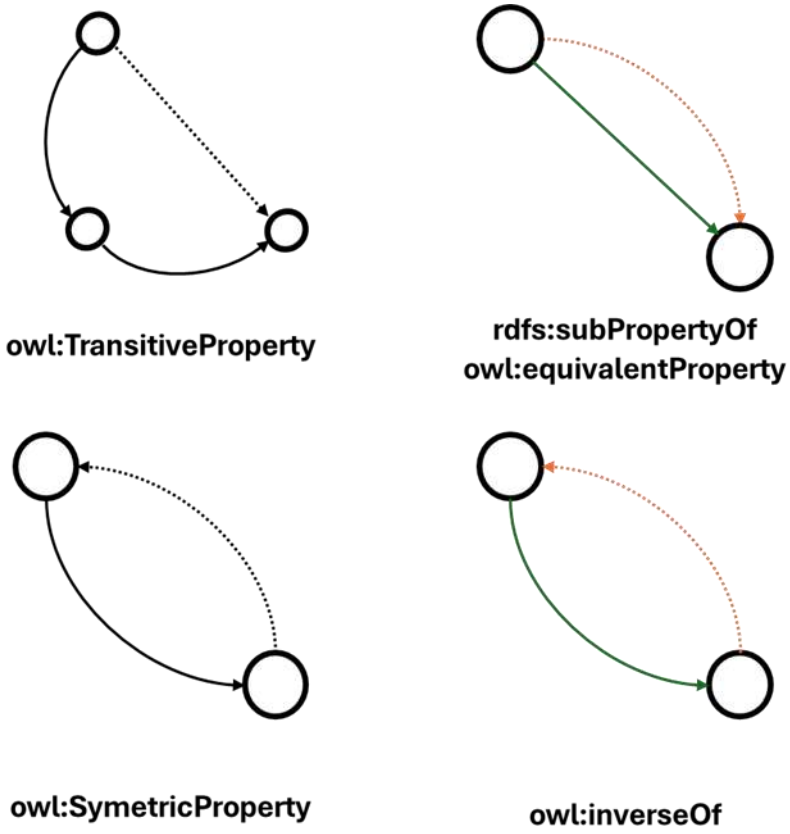
Une fois ce schéma en main, nous pouvons l'appliquer à un graphe RDF grâce à une implémentation que nous décrierons dans la prochaine partie, cela inférera les relations de manière « automatique ».

Malheureusement, RDFS, pour ce qu'on en fait, ne permet pas d'aller plus loin. Nous utiliserons donc OWL, qui encore une fois, comme son nom l'indique, permet de créer des ontologies **[20]**, soit un ensemble de règles simples ou complexes pouvant s'exprimer en logique du premier ordre (dans notre cas). OWL est assez puissant, mais nécessite des moteurs d'inférence **[45]** dédiés qui existent en plusieurs profils **[46]** : OWL2-EL qui est optimisé pour les cas disposant d'un grand nombre de classes ou de propriétées, OWL2-QL (Query Language) pour un grand nombre d'instances et un requêtage plus rapide et OWL2-RL (Rule Language) où le passage à l'échelle est simplifié et le temps de calcul pour les inférences réduit. Lors du stage, OWL2-RL a été choisi, car était le plus adapté.

Avec OWL, nous nous sommes concentrés dans un premier temps sur quatre propriétés différentes :

- owl:equivalentProperty, qui se traduit en logique du premier ordre par : $x\mathcal{R}_1 y \Leftrightarrow x\mathcal{R}_2 y$, soit la relation de « subsomption bilatérale »[12].
- owl:TransitiveProperty, qui se traduit par : $(x\mathcal{R}y \wedge y\mathcal{R}z) \Rightarrow x\mathcal{R}z$, soit la relation de transitivité.
- owl:inverseOf, qui se traduit par : $x\mathcal{R}_1 y \Leftrightarrow y\mathcal{R}_2 x$, soit la relation inverse.
- owl:SymmetricProperty, qui se traduit par : $x\mathcal{R}y \Leftrightarrow y\mathcal{R}x$, soit la relation symétrique.

Voici une image récapitulant ces relations :



Note : Il est intéressant de noter que comme RDF, RDFS et OWL disposent de plusieurs syntaxes, mais nous en utiliserons une proche de RDF Turtle.

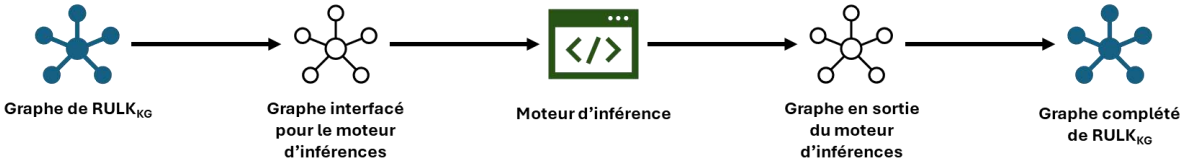
Maintenant que nous avons introduit RDF, RDFS et OWL, nous pouvons essayer de l'appliquer pour les missions du stage.

[12] On désigne cette relation par « subsomption bilatérale » depuis le début, mais si nous étions rigoureux, nous parlerions plutôt de relations d'équivalence.

D.2 – Application du web sémantique dans le cadre du stage

Pour commencer, rappelons que le cadre existant est codé en Python. Il faut donc trouver un moyen d'appliquer les propriétés ci-dessus aux graphes générés par $RULK_{KG}$ de manière interfaçable pour pouvoir en récupérer les résultats en Python plus tard.

Grossièrement, les étapes prévues pour l'implémentation de ces connaissances implicites sont les suivantes :



Ainsi, nous récupérons le graphe existant dans $RULK_{KG}$, nous le transformons en un graphe que peut comprendre un moteur d'inférence (RDF), nous appliquons les règles via un moteur d'inférence (OWL2-RL), nous récupérons le graphe en sortie, que nous transformons enfin en un graphe compatible avec le cadre existant pour pouvoir l'évaluer de la même manière que les autres.

La première étape consiste donc à trouver un moteur d'inférence pour OWL et RDFS compatible avec Python. Dans le cours de l'INRIA mentionné plus tôt, ils nous présentent un logiciel nommé « Corese » **[47]**, développé par eux-mêmes, permettant de manipuler des graphes RDF, RDFS, OWL et d'y effectuer des requêtes SPARQL. Après quelques recherches, j'apprends l'existence d'une version compatible avec Python **[48]** (bien qu'en version bêta).

Cette dernière n'est pas installable avec le manager de package Python PIP, mais ressemble plus à une application Java avec lequel nous utilisons un « gateway » Py4J **[49]** en Python, soit une sorte de pont permettant d'interagir avec des objets/classes Java en Python. Malheureusement, bien que ce que je veuille faire est disponible dans d'autres versions de Corese (Java, Server et Gui), cela ne l'est, après vérification du code Java de l'application, pas dans la version Python. Je dois donc trouver un autre moyen.

Après d'autres recherches, je prends connaissance de l'existence des librairies RDFLIB **[50]** et OWLRL **[51]** en Python, la première permet la création, l'interprétation, la manipulation et le requêtage de graphes RDF, RDFS et OWL, alors que la deuxième, surcouche de la première, permet d'inférer des connaissances usant de OWL et/ou RDFS à partir du graphe de RDFLIB.

Voici donc les étapes du procédé pour inférer les nouvelles connaissances :

1. Nous récupérons le graphe de $RULK_{KG}$ en copiant `KB.relation` contenant les relations du graphe.
2. Nous transformons ces relations en graphe RDFLIB à l'aide de la fonction `KBtoGR`, présente dans le fichier `GraphReasoner.py`, qui ajoute chaque relation une à une de l'objet copié précédemment à l'intérieur du graphe RDFLIB, avec la méthode préexistante `add` de RDFLIB permettant d'ajouter un triplet à un graphe.

3. Nous ajoutons les règles que nous avons construites au sein du schéma RDFS/OWL à l'intérieur du graphe des connaissances à l'aide d'une simple fusion des graphes ($G_1 = G_1 + G_2$ dans le code) *.
4. Nous utilisons OWLRL pour inférer les connaissances en utilisant les règles implantées dans le graphe RDF**.
5. Nous retirons les règles du graphe de connaissances pour éviter que celle-ci soit confondu lors de l'évaluation comme des connaissances ciblées par l'utilisateur. Puis nous exportons le graphe vers $RULK_{KG}$ en utilisant la fonction `GR.GRtoKB` qui permet de remettre les entités et les relations dans le bon format*.

Explication du formatage (*) : Pour des raisons de formatage d'URI dans les schémas RDFS/OWL, les relations décrites sont dans le format : <MajusculeALaPremiereLettreDeChaqueRelationDecrite>. Ainsi, nous faisons un prétraitement réversible des triplets avant l'ajout de ces derniers dans le graphe RDFLIB, pour que ces triplets puissent inférer les règles créées.

Explication des règles (**) : Pour l'étape 4, il faut donc pour appliquer des règles, déjà avoir des règles. Ainsi, après avoir récupéré les relations présentes dans REBEL sur le GitHub comme plus tôt dans le rapport, nous les avons transformées dans un schéma RDFS/OWL. Dans ce dernier, nous avons ensuite appliqué des propriétés aux relations à « vue d'œil », c'est-à-dire qu'au lieu d'avoir une liste exhaustive de couples de relations pour statuer si les membres du couple se subsument, sont inverses, etc., nous avons une liste de l'ensemble des relations avec lesquelles nous essayons de créer des liens entre elles. Voici un extrait du contenu d'un schéma RDFS/OWL utilisé pour ce stage :

```
[…]
<Ancestor> a rdf:Property, owl:TransitiveProperty ; # Created
    rdfs:comment "Created as a super-property of Parent to have a […]"@en ;
    owl:inverseOf <Descendant> .

<Child> a rdf:Property ;
    owl:inverseOf <Parent> ;
    rdfs:subPropertyOf <Descendant> .

<Descendant> a rdf:Property, owl:TransitiveProperty ; # Created
    rdfs:comment "Created as a super-property of Child to have a […]"@en ;
    owl:inverseOf <Ancestor> .

<Parent> a rdf:Property ; # Created
    rdfs:comment "Created over REBEL as an inverse property of child."@en ;
    owl:inverseOf <Child> ;
    rdfs:subPropertyOf <Ancestor> .

<Spouse> a rdf:Property, owl:SymmetricProperty ;
    rdfs:seeAlso <Family> .
[…]
```

Les relations sont représentées de manière unitaire, sur lesquelles nous attachons des propriétés pour les rattacher à d'autres relations (c'est un peu comme si nous rajoutions des métadonnées à un fichier ou que l'on paramétrait un objet). Nous pouvons également créer des relations supplémentaires n'existant pas dans REBEL de manière simple et lisible. Ainsi, dans cet exemple, la relation « Enfant » est l'inverse de la nouvelle relation créée : « Parent » qui est elle-même subsumée par la relation transitive « Ancêtre ». Enfin, si une personne est mariée, alors son conjoint aussi.

Aussi, il est important de noter que comme dit précédemment, il faut faire des nuances entre instances, objets, classes et plus globalement depuis quoi et vers quoi pointent les relations. Le tout étant assez subtil, par mesure de sécurité, j'ai décidé de retirer l'ensemble des subsomptions entre les propriétés <subclassOf>, <partOf>, <instanceOf>, etc. qui sont des relations extrêmement présentes dans les graphes et qui engendrent une grande variation des résultats en fonction de comment elles sont gérées.

Note : Ici nous avons également les notations rdfs:seeAlso et rdfs:comment qui n'influent absolument pas sur les inférences, mais ne sont là que pour la documentation du schéma.

Finalement, la bascule vers ce type de représentation par rapport au tableur Excel aura bel et bien permis d'augmenter la lisibilité et la flexibilité du cadre tout en diminuant la complexité, ainsi qu'en partie le temps de calcul grâce à l'optimisation des fonctions d'inférence.

Note : Retirer les subsomptions entre les relations précitées diminue fortement le temps de calcul en raison d'une réduction drastique du nombre d'inférences engrangées de manière globale.

D.3 – Résultats avec web sémantique et limitations

Le principal avantage du processus mis en place pour le web sémantique est que celui-ci renvoie des graphes compatibles avec l'estimateur θ du cadre $RULK_{KG}$. Cela permet de les évaluer et de les comparer de manière simple avec les résultats antérieurs. Comme pour les autres résultats, vous pourrez les trouver dans le tableur Excel **[34]**.

Note 1 : Depuis la dernière évaluation, une erreur a été découverte dans le fichier des données cibles. Ainsi, nous avions le triplet : {'head': 'Psyllium', 'type': 'instance of <triplet>ispagula husk', 'tail': 'Dietary fiber'}, qui avait été mal généré. Ainsi, il a été remplacé par : {'head': 'Psyllium', 'type': 'instance of', 'tail': 'Dietary fiber'}[13]. Cette action résulte donc en une hausse dans la corrélation.

[13] À la fin du stage, d'autres erreurs du même type ont été découvertes et sont sûrement dues à un problème lors de l'extraction des triplets par REBEL. Ces nouvelles erreurs (au nombre de trois), n'ont pas été supprimées, car le temps manquait pour relancer tous les calculs. Cela ne devrait pas avoir un impact trop fort, car contrairement à celle soulevée ci-dessus, ces nouvelles erreurs sont présentes dans un nombre **très** limité de **documents** consultés par les utilisateurs (3-4/947)

Note 2 : Dans les résultats du tableur Excel, vous trouverez plusieurs résultats : « Same as Study » où la problématique en **Note 1** n'est pas résolue, « Better Res. » où c'est résolu, ainsi que des résultats contentant la mention « PatchLocationVX » ou « LocationPatchVX ». Cette dernière est due à une erreur initiale dans la désignation des propriétés de la relation « Location » et indique que celle-ci a été corrigée.

Voici donc les résultats (en utilisant LocationPatchV0 et en ayant résolu le problème en **Note 1**) :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KW)	0,3022	**0,3086**	N/A
RULK (LM)	0,2955	0,2923	N/A
RULK (NE)	0,0931	0,1180	N/A
RULK (KG, étude)	0,2651	0,2362	N/A
RULK (KG, référence)	0.2691	0,2401	N/A
RULK (KG, RDFS & OWL, utilisateur et cible)	**0,3063**	0,2802	50 minutes
RULK (KG, RDFS, utilisateur et cible)	0,3015	0,2689	35 minutes
RULK (KG, OWL, utilisateur et cible)	0,2983	0,2696	50 minutes

Avec ce tableau, nous observons plusieurs choses :

1. L'intégration des connaissances implicites via RDFS et OWL ont porté leurs fruits au niveau des résultats. En effet, contrairement aux résultats précédents, nous observons une hausse sur l'ensemble des corrélations en ALG et en RPL par rapport à $RULK_{KG}$. Cette hausse est même suffisante pour que l'une des nouvelles versions ait un résultat supérieur à celui de $RULK_{KW}$.

   Comparés à l'étude, nous observons que l'ensemble des modifications qui ont été faites ont permis une amélioration moyenne conséquente de la corrélation face à $RULK_{KG}$ d'environ 13,93% en ALG et d'environ 15,54% en RPL. Plus globalement, nous observons des résultats similaires lorsque l'on utilise OWL, RDFS ou les deux en ALG, mais nous observons bien une différence en RPL lorsque l'on utilise OWL et RDFS, où nous constatons une différence de +4% environ par rapport aux autres versions. Cela démontre bien que OWL et RDFS représentent des aspects différents de la connaissance.

2. Le temps de calcul a fortement réduit. Ainsi, là où avant les temps de calcul dépassaient fréquemment les 2 heures et même 15 heures et 40 minutes pour les subsomptions seules, nous sommes maintenant à des temps bien plus raisonnables en dessous de 1 heure. Par exemple, en prenant les inférences les plus complètes possibles (RDFS & OWL), nous observons une diminution du temps de calcul de presque **95%** par rapport aux subsomptions seules dans l'autre modèle.

À partir de là, nous pouvons déjà dire que le changement vers une représentation RDFS/OWL a dépassé nos attentes. Néanmoins, nous pouvons aller plus loin. En effet,

comme supposé au long de ce rapport, les versions de RULK peuvent être **combinées** afin d'avoir une représentation des connaissances prenant en compte le plus de ces facettes.

Pour fusionner les versions de RULK, nous utiliserons la même méthode présentée dans **[04]**. C'est-à-dire en utilisant la formule :

$$\theta_{RULK\,comb} = \sum_i \omega_i \theta_{RULK_i}$$

Où $\theta_{RULK_{comb}}$ est l'estimateur de la corrélation en combinant les versions de RULK, les i sont des implémentations distinctes de RULK, θ_{RULK_i} l'estimateur de la corrélation pour l'instance de RULK et ω_i le poids optimisé de chaque version de telle manière à ce que : $\sum_i \omega_i = 1$.

Ainsi, en appliquant cette formule, nous obtenons ces nouveaux résultats (dans les mêmes conditions que les précédentes) :

Base Model	ALG	RPL
$RULK_{KG} + RULK_{NE}$	0.30400	0.29040
$RULK_{KG} + RULK_{LM}$	0.32930	0.30310
$RULK_{KG} + RULK_{KW}$	0.31650	0.32250
$RULK_{KG} + RULK_{KW} + RULK_{LM}$	0.33320	0.33330
$RULK_{KG} + RULK_{NE} + RULK_{LM}$	0.35020	0.34752
$RULK_{KG} + RULK_{NE} + RULK_{KW}$	0.35465	0.35550
$RULK_{KG} + RULK_{NE} + RULK_{KW} + RULK_{LM}$	0.36044	0.36163
$RULK_{KG+(RDFS)}$	ALG	RPL
$RULK_{KG+} + RULK_{NE}$	0.33297	0.32080
$RULK_{KG+} + RULK_{LM}$	0.33121	0.31245
$RULK_{KG+} + RULK_{KW}$	0.34633	0.33377
$RULK_{KG+} + RULK_{KW} + RULK_{LM}$	0.34742	0.33530
$RULK_{KG+} + RULK_{NE} + RULK_{LM}$	0.36574	0.35787
$RULK_{KG+} + RULK_{NE} + RULK_{KW}$	0.37357	0.36884
$RULK_{KG+} + RULK_{NE} + RULK_{KW} + RULK_{LM}$	**0.37571**	0.37165
$RULK_{KG+(OWL)}$	ALG	RPL
$RULK_{KG+} + RULK_{NE}$	0.33050	0.32543
$RULK_{KG+} + RULK_{LM}$	0.32970	0.31808
$RULK_{KG+} + RULK_{KW}$	0.34476	0.33434
$RULK_{KG+} + RULK_{KW} + RULK_{LM}$	0.34603	0.33584
$RULK_{KG+} + RULK_{NE} + RULK_{LM}$	0.36078	0.35537
$RULK_{KG+} + RULK_{NE} + RULK_{KW}$	0.36870	0.36647
$RULK_{KG+} + RULK_{NE} + RULK_{KW} + RULK_{LM}$	0.37127	0.36960
$RULK_{KG+(RDFS/OWL)}$	ALG	RPL
$RULK_{KG+} + RULK_{NE}$	0.33845	0.32543
$RULK_{KG+} + RULK_{LM}$	0.33427	0.31808
$RULK_{KG+} + RULK_{KW}$	0.34884	0.33880
$RULK_{KG+} + RULK_{KW} + RULK_{LM}$	0.34938	0.33989
$RULK_{KG+} + RULK_{NE} + RULK_{LM}$	0.36548	0.36054
$RULK_{KG+} + RULK_{NE} + RULK_{KW}$	0.37315	0.37148
$RULK_{KG+} + RULK_{NE} + RULK_{KW} + RULK_{LM}$	0.37537	**0.37401**

À partir de ces tableaux, nous observons déjà que l'ensemble des combinaisons utilisant

$RULK_{KG+}$ (soit la nouvelle version), engendre de meilleurs résultats que leurs équivalents utilisant $RULK_{KG}$, et ce que ce soit avec OWL, RDFS ou les deux. Si nous regroupons les résultats, nous observons une hausse moyenne dans la corrélation en RPL de 3,41% pour OWL, 3,88% pour RDFS et 5,10% pour RDFS/OWL. Une tendance similaire se dégage en ALG avec des hausses de 4,52% pour OWL, 5,70% pour RDFS et 5,95% pour les deux.

Au niveau des implémentations, les combinaisons utilisant RDFS et OWL ont généralement de meilleures performances comparées à OWL ou RDFS seuls si l'on regarde la corrélation en RPL. Ce résultat est cependant à nuancer, car les corrélations en ALG sont très proches. Néanmoins, nous pouvons toujours confirmer ce qui était dit plus tôt à propos des différents aspects de la connaissance que portraient RDFS et OWL. Nous pouvons également rajouter que parmi les connaissances implicites inférées, la propriété de subsomption (présente dans RDFS) semble avoir un poids prépondérant dans les résultats, notamment pour la corrélation en ALG.

Cependant, nous remarquons que lorsque que $RULK_{LM}$ est impliqué dans une combinaison de trois éléments où moins utilisant $RULK_{KG+}$, une augmentation plus lente de la corrélation est observée. Ainsi, les combinaisons l'utilisant disposent d'une augmentation de leur corrélation en moyenne inférieure de 56,75% en RPL et de 63,86% en ALG par rapport aux combinaisons qui ne l'utilisent pas. Cela pourrait supposer que le type de connaissances extrait par le modèle de langage utilisé dans $RULK_{LM}$[14] serait en partie similaire aux connaissances inférées, ce qui entrainerait une progression plus lente, car nous décrierions deux fois le même aspect des connaissances.

Enfin, comme pour $RULK_{KG}$, l'ajout d'un modèle dans les combinaisons de $RULK_{KG+}$ entraîne toujours une amélioration des performances, ce qui prouve bien que ce modèle représente toujours un aspect, non abordé précédemment, des connaissances.

Note : Comme espéré lorsque l'on essaie la fusion de l'ensemble des versions KG, NE, LM, KW et KG+, nous obtenons les mêmes résultats que la fusion de KG+, NE, LM et KW. Cela veut dire que la nouvelle version ne floute pas les connaissances que nous avions déjà.

Limitations : Comme tous les résultats, ces derniers ont des limites. Ainsi, le cadre RULK lui-même ne permet ni d'évaluer le gain de connaissance par rapport à la forme (ne peut pas tirer de gain à partir des images, vidéos, etc.), ni par rapport au comportement de l'utilisateur (nombre de requêtes, temps passé, traçage du mouvement des yeux pour savoir quel texte a été lu, etc.). Or, d'après Yu et al, dans leur article **[52]**, nous observons que ces données annexes peuvent être d'une grande utilité dans la prédiction du gain de connaissances de l'utilisateur. Un mix de ces deux systèmes pourrait donc être une bonne piste d'amélioration future.

Aussi, dans le cadre de ce stage, nous n'avons mis en place, par souci de simplicité, que cinq propriétés de base. Cependant, comme dit plus tôt, ces propriétés ne permettent

[14] Nous observons, en réalité, après l'application de l'**Erratum** l'exact inverse, les combinaisons avec LM engendreraient de bien meilleurs résultats que les combinaisons sans LM, mais ces résultats restent à confirmer.

pas de capturer des connaissances implicites complexes (comme l'interprétation de chaînes de relations distinctes par exemple). De futures améliorations pourraient donc se voir à ce niveau-là, pour la création de dictionnaires d'ontologies applicables à RULK (voir partie **Bonus**).

De plus, d'autres techniques pour compléter les graphes pourraient être explorées, en passant par la reconnaissance du type des entités pour ainsi appliquer des règles sur des classes en plus des propriétés, la mise en place de propriétés contraignantes dans OWL comme le fait qu'on ne puisse pas être son propre parent, a priori, pour introduire une sorte « **d'esprit critique** » au sein du cadre RULK, l'utilisation de graphes de référence, etc.

Enfin, il y a la problématique, plus globale, qui consiste à savoir quelle est la connaissance cible. En effet, rien ne nous dit que la cible de l'utilisateur est l'article Wikipédia, il faut donc trouver un moyen d'introduire de l'incertitude dans les résultats (travail en cours par M. Nasser). D'un autre côté, nous ne pouvons jamais être sûrs à 100% de ce que l'utilisateur « connaît » parmi ce que l'on extrait des documents consultés. Ainsi, l'introduction d'une logique floue **[53]** pourrait aider, après tout, les connaissances dans un domaine ne sont pas que « je connais tout » ou « je ne connais rien », mais prennent aussi l'ensemble des valeurs intermédiaires.

E – Processus de rédaction scientifique et publication

Les résultats étant au-dessus de ceux proposés précédemment par $RULK_{KG}$ et moyennant l'implémentation de connaissances implicites des utilisateurs sur les relations dans l'estimation du gain de connaissance de l'utilisateur, dans le domaine de la recherche en tant qu'apprentissage, j'ai participé à la rédaction d'un article scientifique en tant qu'auteur principal avec M. Hadi Nasser, Mme. Célia Da Costa Pereira, Mme. Cathy Escazut et M. Andréa Tettamanzi, tous en affiliation avec le laboratoire I3S.

Dans cette partie, je vous raconte le déroulement global qui a permis sa création.

Pour commencer, avant de publier un article, il faut trouver une conférence à qui le soumettre. Dans notre cas, la rédaction de l'article a commencé le 16 mai, à ce moment-là, il ne reste donc plus que la **C**onference on **I**nformation and **K**nowledge **M**anagement **CIKM 2024 [54]**, dans le domaine de la recherche en tant qu'apprentissage, qui nous permettrait de soumettre l'article à une date proche (3 juin 23h59 AoE).

Ainsi, M. Nasser m'a introduit à Overleaf **[55]**, un éditeur LaTeX en ligne, afin de pouvoir rédiger l'article dans le format demandé par la conférence. Soit un format double colonnes, 4 pages maximum, avec références illimitées.

L'article **[56]** en lui-même est intitulé : « Enhancing User's Knowledge Gain Estimation in Search-as-Learning Using Implicit Knowledge », soit en français : « Renforcement de l'estimation du gain de connaissance de l'utilisateur dans la recherche en tant qu'apprentissage en utilisant des connaissances implicites ». Et est référencé par les mots clés « Knowledge Tracking », « Search-As-Learning », « Knowledge Graphs » et « Information Retrieval ».

Il est structuré en cinq parties distinctes :

- **L'introduction et les travaux liés** décrivent dans un premier temps les connaissances de la littérature sur la recherche en tant qu'apprentissage, la problématique que l'on rencontre et ce que l'on a fait.
- **Le « Background »**, cela inclue la présentation du cadre RULK et de ses versions avec plus de précisions sur $RULK_{KG}$ que nous étendons.
- **$RULK_{KG+}$, une extension de $RULK_{KG}$**, où plus précisément la définition des « connaissances implicites » utilisées, ainsi qu'une description de l'implémentation faite.
- **Expériences et résultats**, où l'on décrit le jeu de données utilisé et où l'on détaille la définition de l'ALG et du RPL, les résultats des modèles seuls et combinés.
- **Conclusion**, c'est ici que l'on résume l'article et que l'on donne des pistes d'amélioration.

Concernant l'abstract, une sorte de résumé de l'article, celui-ci suit la trame :

- **Contexte** (Recherche en tant qu'apprentissage),
- **Objectif** (Estimer le gain de connaissance de l'utilisateur),
- **Littérature** (Représentation sous forme de graphes de connaissances),
- **Problématique** (La non-mention des connaissances « évidente » dans les documents lus par l'utilisateur),
- **Solution** (Inférer les connaissances implicites),
- **Résultat** (Amélioration des résultats et prise en compte d'un aspect additionnel de la connaissance).

Et le voici (en Anglais) :

> In the context of search as learning, users achieve their learning goals (target knowledge) through exploratory research. Consequently, it is essential to track the user's knowledge state in order to estimate how close the user is to achieving his/her learning goals.
>
> A recently proposed approach uses knowledge graphs to represent both the user's knowledge and the user's target knowledge. However, some basic pieces of information are omitted because they are considered too self-evident to be included in resources during the user's search session. This omission results in incomplete relations between resources and reduces the user's knowledge gain estimation. Our approach, which is based on this foundation, introduces a method for enhancing the user's knowledge gain estimation by considering implicit knowledge. In more precise terms, our approach extends the representation of both the user's and the target knowledge by utilising different predicate properties to complete these relations. We show that the newly proposed approach improves the knowledge gain estimation by taking into account an additional aspect of knowledge.

L'abstract ci-dessus a donc été envoyé le 27 mai (UTC+2), l'article a lui été envoyé à la conférence le 4 juin avant 13h59 (UTC+2). À date d'écriture du rapport de stage, je ne sais pas si l'article passera les Reviewer et sera sélectionné, après tout, la sélection d'un article se fait avec : « 50% de chances, 25% de qualité et 25% de je ne sais pas » – Un membre du laboratoire I3S. Si l'article est sélectionné, une notification nous sera envoyée le 16 juillet à 23h59 (AoE) et nous devrons soumettre la version définitive de l'article le 8 août 2024. Dans ce cas, au moins un des auteurs devra se rendre à Boise aux États-Unis entre le 21 et le 25 octobre 2024 pour présenter le travail qui a été fait.

V – Bonus : Ontologies pour la complétion des graphes

Dans cette partie, seront abordées les ontologies pour la complétion des graphes, soit l'une des pistes d'amélioration énoncées dans l'article. Il s'agira ainsi d'une nouvelle mission confiée à la fin du stage, étant donné la complétion des missions précédentes, qui aura pour objectif de définir des ontologies complexes et de les évaluer.

A – Définition des ontologies et premières ontologies complexes

Tout au long du rapport, nous avons parlé d'ontologies pour compléter des graphes, fusionner des entités, etc. Mais, nous n'avons nullement défini le terme d'ontologie de manière « propre ».

Pour commencer, le mot « ontologie », du latin « ontologia », nous vient de la philosophie et désigne par son étymologie, onto- (étant, ce qui est) et -logia (discours, traité), l'étude de l'être, des attributs et de son essence **[57]**. Ce mot a commencé à se populariser dans le domaine de l'informatique suite aux travaux de Tomas R. Gruber en 1993 **[58]**. Ce dernier a pour objectif d'uniformiser les vocabulaires pour définir des ontologies, afin, déjà, de représenter des connaissances partagées dans le cadre de systèmes d'IA.

En informatique, le sens de ce mot a légèrement dérivé de son étymologie philosophique et représente aujourd'hui : « un document ou un fichier qui définit formellement les relations entre termes. Le cas le plus typique d'une ontologie pour le Web étant une taxonomie et un ensemble de règles d'inférence » – Tim Berners-Lee, 2001 **[59]**.

Pour notre part, nous nous concentrerons sur un type d'ontologie portant sur les relations et plus particulièrement l'inférence de relations à partir de chaînes de relations, soit les fameuses « règles plus complexes » que j'évoquais au début du rapport. Par exemple si un enfant A possède un parent B appartenant à la famille C, alors l'enfant A appartient à la famille C.

Cela peut se faire en OWL avec la propriété owl:propertyChainAxiom, qui s'utilise ainsi (en Turtle) :

```
<BasinCountry> a rdf:Property ;
    owl:propertyChainAxiom (<Tributary> <BasinCountry>) .

<InstanceOf> a rdf:Property ;
    owl:propertyChainAxiom (<InstanceOf> <SubclassOf>) .
```

Plus concrètement, nous déclarons ici, que si une entité est un affluent d'une autre entité et que celle-ci se situe dans un bassin fluvial, alors la première entité se situe aussi dans le bassin fluvial. De même, si une entité est une instance d'une sous-classe, alors elle est aussi une instance de la classe mère. Plus globalement, nous déclarons des chaînes en alignant les relations les unes derrière les autres dans les parenthèses.

Ces relations sont définies dans le même schéma RDFS/OWL que précédemment et sont interprétées de la même manière, c'est-à-dire qu'il n'y a pas besoin de retoucher au code pour observer une modification des résultats. Une partie des ontologies mises en place sont décrites dans le tableur Excel « OntologyRepository.xlsx » **[60]**.

Le principal problème lié à ces ontologies est qu'elles sont longues à trouver. En effet, comme expliqué précédemment, nous essayons de trouver des connaissances **implicites chaînées**. Ainsi, là où il est simple de trouver des connaissances implicites sur un niveau, trouver des chaînes entières pour expliquer une relation demande beaucoup plus d'abstraction. De plus, là où il aurait été possible pour d'autres problèmes de recourir à une approche quasi exhaustive pour trouver les meilleurs schémas, ce n'est pas le cas ici en raison du temps de calcul beaucoup trop élevé (50 minutes/version) par rapport au nombre de propriétés à essayer sur les relations.

Note : Il est intéressant de savoir qu'il existe une branche de l'ingénierie spécialisée sur ce type de tâches complexes liées aux connaissances : « l'ingénierie des connaissances ».

Après avoir créé différentes ontologies, voici les résultats engendrés :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, article)	0.2651	0,2362	N/A
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes
RULK (KG, OntologiesV1 RDFS & OWL)	**0,3107**	**0,2903**	N/A ~1 heure
RULK (KG, OntologiesV2 RDFS & OWL)	0,3094	0,2869	1 heure
RULK (KG, OntologiesV1-2 RDFS & OWL)	0,3062	0,2827	57 minutes
RULK (KG, OntologiesV0 RDFS & OWL)	0,3053	0,2803	50 minutes
RULK (KG, SelectedOntologiesV0 RDFS & OWL)	0,3058	0,2802	57 minutes
RULK (KG, SelectedOntologiesV1 RDFS & OWL)	0,3033	0,2759	58 minutes
RULK (KG, AddOnV0 RDFS & OWL)	0,3006	0,2757	52 minutes

Globalement, nous pouvons observer que les résultats en appliquant les ontologies sont assez disparates. En effet, même si une règle nous semble « logique », celle-ci ne peut l'être qu'en apparence. De plus, comme nous testons plusieurs dizaines de règles à la fois, il est compliqué de savoir lesquelles ont un impact positif ou négatif sur les corrélations, même si cela nous permettrait de sélectionner les règles les plus pertinentes.

B – Évaluation des ontologies

Une solution pour tester les ontologies serait donc de réduire de manière significative le temps de calcul, afin d'évaluer les impacts sur les corrélations en

rajoutant les règles une à une. Cela n'est cependant pas possible avec l'architecture actuelle. Nous allons donc légèrement la modifier de manière que cela soit possible. Voici les modifications qu'il faudrait faire :

- **Séparer le schéma RDFS/OWL en plusieurs schémas contenant 1 règle.** En effet, cela permettra de savoir quelles règles ont un impact sur la corrélation. De plus, nous devons procéder ainsi, car les règles « complexes » sont en réalité encodées en plusieurs triplets dans RDFLIB, ce qui nous empêche de simplement itérer sur le graphe RDFS/OWL (cela ne testerait que des « parties » de règles au lieu de règles complètes).
- **Créer un sous-échantillon pour tester les règles.** Comme dit plus tôt, il serait trop long de tester pour l'ensemble des individus de l'échantillon. Nous allons donc créer un sous-échantillon de l'échantillon initial.
- **Pour chaque sous-échantillon : appliquer les règles.**

Note : Une limite de cette démarche est que les règles ne sont pas toutes indépendantes. En effet, si l'on déclare que la relation « Localisé dans un territoire administré » est une sous-propriété de « Localisé dans » qui est une sous-propriété de « Localisation ». Alors, nous n'aurons pas l'ensemble des implications. C'est-à-dire que les résultats ne prendront pas en compte l'ensemble de l'écosystème de ces règles, mais uniquement une partie de celui-ci. Une autre limite résulte de l'impossibilité de trouver les règles qui, quand sont gérées ensemble engendrent un moins bon résultat par conflit. Cela reste néanmoins un premier pas pour déceler les règles candidates.

B.1 – Construction des échantillons

Pour mettre en place ces étapes, nous avons créé un script prenant en entrée un schéma RDFS/OWL et rendant en sortie autant de schémas RDFS/OWL qu'il y a de règles dans le schéma en entrée, vous le trouverez sous le nom Separator.py.

À partir de là, nous devons créer un sous-échantillon de l'échantillon initial. Dans l'idéal, il nous faudrait une marge d'erreur inférieure ou égale à 5% avec un taux de confiance de 95%. Nous savons que l'échantillon initial est composé de 127 individus représentés par 7 sujets distincts répartis ainsi :

Sujet	Nombre d'individus	Fréquence des individus
Considérations sur la datation au carbone (C)	*22*	*17,32%*
L'éthique (E)	*20*	*15,75%*
La crise des Subprimes (Sub)	*20*	*15,75%*
La perte de l'audition due au bruit (PAB)	*19*	*14,96%*
Le cycle économique (CE)	*16*	*12,60%*
Le syndrome de l'intestin irritable (SII)	*15*	*11,81%*
Les OGM (OGM)	*15*	*11,81%*
Total	**127**	**100%**

Grâce à cela, nous pouvons trouver la taille du sous échantillon à partir de la formule de la marge d'erreur à 5% d'une proportion (majorée) ainsi :

$$1.96\sqrt{\frac{1-f}{n}\hat{p}(1-\hat{p})} \leq 0.98\sqrt{\frac{1-f}{n}} \leq 0.98\sqrt{\frac{1}{n}-\frac{1}{N}}$$

Il nous faut donc trouver n à partir de la formule ci-dessus, ce qui nous donne une fois calculer : $n \geq 96$. Or, cela pourrait représenter un temps de calcul conséquent. Nous allons donc créer plusieurs plus petits sous-échantillons afin de diminuer le temps de calcul. Ainsi plus nous aurons de sous-échantillons dans un temps « raisonnable », plus notre marge de confiance sera réduite.

Après réflexion, j'ai décidé d'aller jusqu'à 7 sous-échantillons (en espérant y arriver), de 14 individus chacun, contenant 2 individus par sujet (cela nous donnerait une stratification proportionnelle sur 1 sous-échantillon par rapport à l'échantillon initial) :

Sujet	Nombre d'individus 1 sous-échantillon	Nombre d'individus 7 sous-échantillons	Fréquences initiales de l'échantillon
C	*2 (14,29%)*	*14 (11,02%)*	*17,32%*
E	*2 (14,29%)*	*14 (11,02%)*	*15,75%*
Sub	*2 (14,29%)*	*14 (11,02%)*	*15,75%*
PAB	*2 (14,29%)*	*14 (11,02%)*	*14,96%*
CE	*2 (14,29%)*	*14 (11,02%)*	*12,60%*
SII	*2 (14,29%)*	*14 (11,02%)*	*11,81%*
OGM	*2 (14,29%)*	*14 (11,02%)*	*11,81%*
Total	**14 (~100%)**	**98 (77.17%)**	**100%**

Au niveau de la création des sous-échantillons, il s'agira d'une mission confiée à une nouvelle fonction de la classe SC. Celle-ci se chargera de créer les sous-échantillons de manière que le nombre d'utilisateurs par sujet soit d'exactement 2 dans chaque sous-échantillon et que ces derniers soient indépendants les uns-des-autres (c'est-à-dire qu'un individu ne soit pas dans plusieurs sous-échantillons).

La fonction permettra également d'évaluer automatiquement les schémas RDFS/OWL indépendamment des précédents et enregistrera les résultats du calcul de similarité pour chaque règle (la fonction est dérivée de celle du calcul de similarité de base).

B.2 – Détection des règles « pertinentes »

Par chance, le temps de calcul pour la mise en place des sept échantillons et le test des règles sur ces derniers a été plus faible qu'escompté. En effet, cela a été calculé en environ 56 heures pour les 98 règles utilisées pour les données de l'article ainsi que les 43 règles créées par la suite (total : 141 règles). Ce qui nous donne environ 24 minutes par règle en moyenne.

Ainsi, nous pouvons regrouper les 7 sous-échantillons précédemment créés, en un seul grand sous-échantillon comprenant 98 individus afin de faire un sondage post-stratifié.

Mais, avant cela, il nous faut une méthodologie pour savoir si une règle est « pertinente » à elle seule. Pour cela, nous utilisons la fonction `dataTransform` du script `SampleFunction.R`

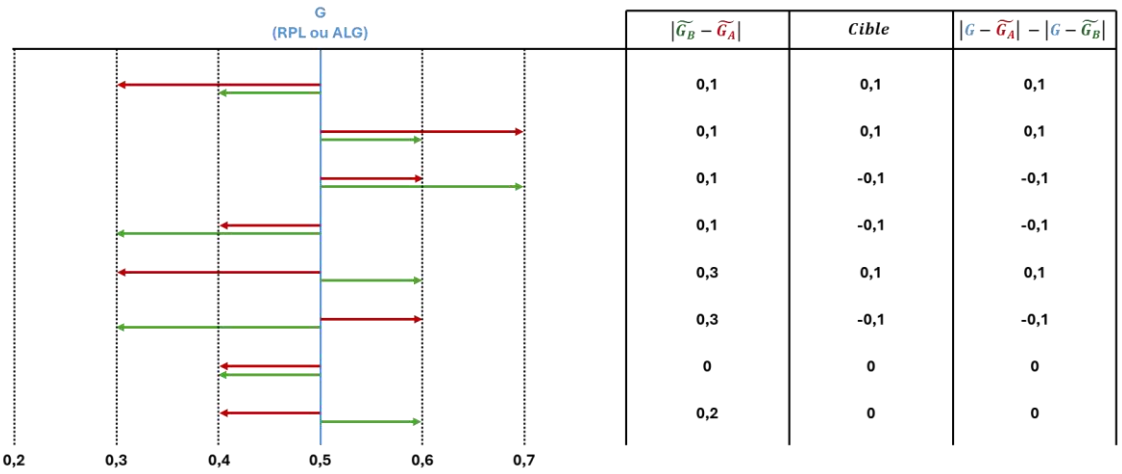
pour transformer les données en sortie de la fonction test_Ontologies[15] de la classe SC en Python.

Cela nous donne une liste de quatre data frames :

- **GA_estimate :** Un data frame de dimension 97x3[16] contenant l'ID de l'utilisateur et son sujet en plus de l'estimation du gain de connaissance avec l'ontologie vide (sans règles).
- **GB_estimate :** Un data frame de dimension 97x45 (avec règles supplémentaires) ou 97x100 (avec règles de base), contenant aussi l'ID et le sujet, mais également l'ensemble des gains de connaissances estimés pour chaque règle.
- **ActualG :** Un data frame de dimension 97x45 ou 97x100, contenant la même chose que le précédent à l'exception qu'au lieu que ce soit l'estimation du gain de connaissance pour chaque règle, il s'agit du gain de connaissance réel (même valeur pour chaque règle).
- **DistanceGain :** Sûrement le plus important pour le moment, celui-ci nous permet de connaître le « gain » de distance entre la nouvelle estimation et celle de l'ontologie vide par rapport aux gains de connaissances réels. Si la valeur est positive, cela veut dire que l'on se « rapproche » du gain de connaissance réel, alors qu'une valeur négative veut dire que l'on s'en éloigne.

La formule du gain de distance a été inférée à partir d'observations, voici donc sa formule et un exemple :

$$\left|G - \widetilde{G_A}\right| - \left|G - \widetilde{G_B}\right|$$



$\left\|\widetilde{G_B} - \widetilde{G_A}\right\|$	*Cible*	$\left\|G - \widetilde{G_A}\right\| - \left\|G - \widetilde{G_B}\right\|$
0,1	0,1	0,1
0,1	0,1	0,1
0,1	-0,1	-0,1
0,1	-0,1	-0,1
0,3	0,1	0,1
0,3	-0,1	-0,1
0	0	0
0,2	0	0

Ici, G représente le gain de connaissance « réel », l'ALG ou le RPL, nous pouvons le retrouver dans ***ActualG***. $\widetilde{G_A}$ Représente le gain de connaissance estimé à partir de

[15] Cette fonction est désormais rangée dans la classe SCRDF, héritant de la classe SC.

[16] A cause d'une erreur d'entrée dans test_Ontologies, le 98ème individu n'a pas été calculé et nous avons donc 97 individus seulement au lieu de 98. La différence se fait sur le sujet PAB. Pour le sondage post-stratifié, cela ne posera pas de problèmes. En effet, la fréquence des modalités dans chaque strate est supérieure à 10%.

l'ontologie vide, nous le retrouvons dans ***GA_estimate***. Enfin, $\widetilde{G_B}$ correspond aux gains de connaissances estimés par règles que l'on retrouve dans ***GB_estimate***.

Note : Dans R, le calcul est fait de manière vectorielle pour obtenir ***DistanceGain***. Cependant, il est tout de même important de mentionner que dans un cadre vectoriel, les éléments de cette formule ne possèdent pas le même nombre de variables. Nous devons donc retirer à chaque colonne de G le vecteur $\widetilde{G_B}$ (une fois que l'on a retiré les informations comme l'ID ou le sujet).

À partir de là, nous allons essayer d'estimer la moyenne du gain de distance de chaque règle à partir d'un sondage post-stratifié. Ainsi, si la moyenne est supérieure à 0 (naturellement avec ses marges de confiance), alors nous considérerons celle-ci comme ayant un impact positif, sinon nous la considérerons comme ayant un impact négatif. Voici l'estimateur de la moyenne d'un sondage post-stratifié :

$$\widehat{\mu_{POST}} = \sum_{h=1}^{H} \frac{N_h}{N} \overline{y_h}$$

Où $\widehat{\mu_{POST}}$ est l'estimateur de la moyenne, h la h-ième strate, H le nombre de strates, N la taille de la population, N_h la taille de la strate et $\overline{y_h}$ la moyenne intrastrate.

Pour la marge de confiance, il nous faut également une estimation de la variance de l'estimateur de la moyenne que nous calculons ainsi (car n grand) :

$$Var(\widehat{\mu_{STPROP}}) = \frac{1-f}{n} s_{intra}^2$$

Où $Var(\widehat{\mu_{STPROP}})$ est une approximation de l'estimation de la variance de la moyenne de l'estimateur, f le taux de sondage, n la taille de l'échantillon et s_{intra}^2 la moyenne des variances corrigées de chaque strate.

Note : Lors du nettoyage des données, certaines règles disposaient d'un $\widetilde{G_B}$ constant au-dessus/en-dessous de 0, alors que les règles ne pouvaient pas s'appliquer, car les conditions n'étaient pas réunies. Chacune des règles impactant différemment le graphe, j'ai décidé d'automatiquement resituer le mode à 0, c'est-à-dire que si une constante apparait, elle sera retransférée en 0, si 0 n'est pas déjà le mode.

Il nous suffit dorénavant d'afficher les règles et leur moyenne à travers un graphique en barres. Voici un des résultats obtenus :

Ici, il n'y a pas besoin de regarder l'ampleur de l'impact, mais juste si c'est positif ou négatif. Avec cela, nous nous rendons compte de trois choses :

1. La grande majorité des règles n'ont pas d'impact important. Cela peut s'expliquer par le fait que les conditions pour que ces règles s'activent ne soient pas réunies dans les graphes des utilisateurs ou qu'il y a eu symétrie parfaite des estimations par rapport aux résultats réels,
2. Certaines sont délétères à notre modèle quand elles sont **seules**,
3. D'autres sont bénéfiques lorsqu'elles sont **seules**.

Ici, nous observons que 8 règles sont bénéfiques :

- **ID58** : Manufacturer -> Creator
- **ID30** : FollowedBy [inverseOf] Follows
- **ID32** : Follows [inverseOf] FollowedBy
- **ID23** : Developer -> Creator
- **ID24** : DiscoverOrInventor -> Creator
- **ID52** : LocatedInTheAdministrativeTerritorialEntity -> LocatedIn
- **ID33** : FoundedBy -> Creator
- **ID25** : Dissolved,AbolishedOrDemolishedDate -> Date

Et que 9 règles ne le sont pas :

- **ID85** : SubclassOf (transitive)
- **ID27** : FacetOf (transitive)
- **ID04** : Author -> Creator
- **ID68** : PartOf (transitive)
- **ID34** : HasEffect (transitive)
- **ID77** : PublicationDate -> Date
- **ID66** : OwnerOf [inverseOf] OwnedBy
- **ID65** : OwnedBy [inverseOf] OwnerOf
- **ID29** : FollowedBy (transitive)

À partir de là, nous pouvons établir des ontologies ne prenant en compte que les règles positives. Voici alors les résultats en intégrant uniquement ces règles plus quelques autres à partir des règles bonus :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, article)	0.2691	0,2401	N/A
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes
RULK (KG, AfterFullAnalysis_RPL RDFS & OWL)	0,3007	0,2748	38 minutes

Ce n'est simplement pas bon, cela est dû comme nous l'avons dit plus tôt que les règles ne sont pas indépendantes. Elles interagissent positivement ou négativement entre elles pour inférer les connaissances de l'utilisateur. Ainsi, il faut tester plutôt que des règles seules, des groupes de règles.

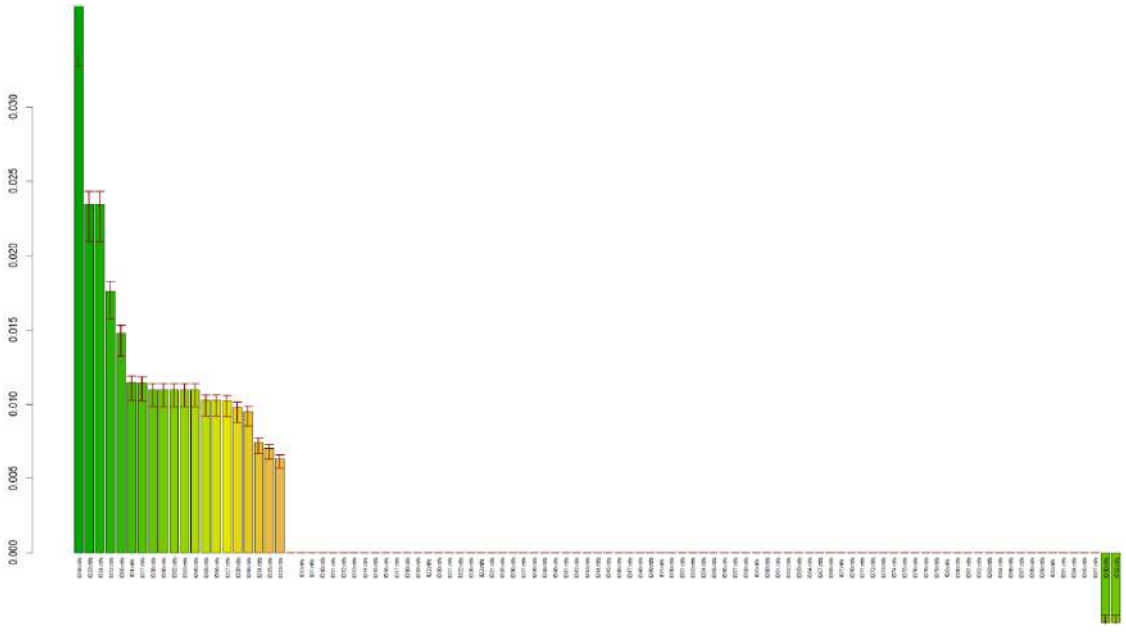
Néanmoins, avant de faire cela, nous avons une autre piste à explorer. En effet, ces résultats pourraient simplement venir de la mesure qui nous permet de sélectionner les règles. Celle-ci, bien que permettant de savoir si nos estimations se rapprochent du RPL ou de l'ALG n'est pas adaptée à la mesure de corrélation que nous cherchons à avoir, car elle ne prend pas en compte la variation. De plus, l'ALG et le RPL ne peuvent pas être directement comparés à l'estimation du gain de connaissances, dans la mesure où les unités et échelles de ces trois dernières sont différentes.

Une autre mesure que nous emploierons est donc la suivante :

$$cor\left(G, \widetilde{G_B}\right) - cor\left(G, \widetilde{G_A}\right)$$

Dans R, nous calculons donc la différence entre les corrélations de Pearson **[61]**, à l'aide de `cor.test` à un seuil de 95% dans la fonction `dataTransform`. Les résultats sont ensuite enregistrés en sortie dans **CorrelationGain**.

En faisant ensuite la même démarche que tout à l'heure, voici un des résultats obtenus :



Nous avons donc bien plus de règles « pertinentes » seules et beaucoup moins qui ne le sont pas, même si nous observons toujours que la grande majorité n'a simplement pas d'impact sur la corrélation.

Au global, voici les règles qui ont un gain unitaire positif sur la corrélation (ici en RPL) : **ID58b** Manufacturer -> Creator **/ ID23b** Developer -> Creator **/ ID24b** DiscoverOrInventor -> Creator **/ ID52b**: LocatedInTheAdministrativeTerritorialEntity -> LocatedIn **/ ID85b** SubclassOf (transitive) **/ ID04b** Author -> Creator **/ ID77b** PublicationDate -> Date **/ ID28b** FieldOfThisOccupation -> FieldOfWork **/ ID90b** Use [inverseOf] UsedBy **/ ID92b** UsedBy [inverseOf] Use **/ ID93b** UsedBy [inverseOf] Uses **/ ID96b** Uses [inverseOf] UsedBy **/ ID65b** OwnedBy [inverseOf] OwnerOf **/ ID66b** OwnerOf [inverseOf] OwnedBy **/ ID27b** FacetOf (transitive) **/ ID29b** FollowedBy (transitive) **/ ID68b** PartOf (transitive) **/ ID34b** HasEffect (transitive) **/ ID25b** Dissolved,AbolishedOrDemolishedDate -> Date **/ ID33b** FoundedBy -> Creator **/ ID16a** HasCause o HasEffect -> HasCause **/ ID34a** ParentOrganization [inverseOf] Subsidiary **/ ID36a** Subsidiary [inverseOf] ParentOrganization **/ ID24a** InstanceOf o SubclassOf -> InstanceOf **/ ID22a** Screenwriter -> Creator.

Note : les ID se terminant par « b » indiquent que ce sont les règles de base, c'est-à-dire celles utilisées pour les données de l'article, alors que celles en « a » sont les règles additionnelles. L'ensemble des résultats pour l'analyse individuelle des règles est disponible en **[62]**.

Cela nous donne finalement ce résultat au niveau de la corrélation :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, article)	0.2691	0,2401	N/A
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes
RULK (KG, AfterCorrelationGainAnalysis_RPL RDFS & OWL)	0,2991	0,2695	38 minutes

Soit des corrélations moins bonnes encore que la métrique précédente. Alors que cette dernière semble avoir plus de sens dans sa construction que la précédente, il est difficile de le démontrer.

La différence entre les deux mesures pourrait tout de même s'expliquer par le fait que les estimations du gain de connaissances et des gains de connaissances réels sont premièrement dans des unités différentes et deuxièmement dans des intervalles différents. Mais cela n'explique pas pourquoi la deuxième mesure est moins bonne que la première.

Nous allons donc essayer cette fois, notre piste initiale avec des groupes de règles comme annoncé précédemment.

B.3 – Groupement de règles

Dans cette partie, nous nous intéresserons donc aux groupes de règles. L'avantage que l'on a pour cette partie par rapport à la précédente est que l'on peut en réalité déjà tester plusieurs règles à la fois. En effet, la fonction permettant de tester les règles lit tous les schémas RDFS/OWL présents dans un dossier. Ainsi, nous pouvons en réalité exécuter la même fonction pour agir sur des groupes de règles distincts, à condition qu'elles soient dans des fichiers différents. Mettre en place de tels groupes nous permettrait d'inférer des connaissances en prenant en compte la non-indépendance des règles.

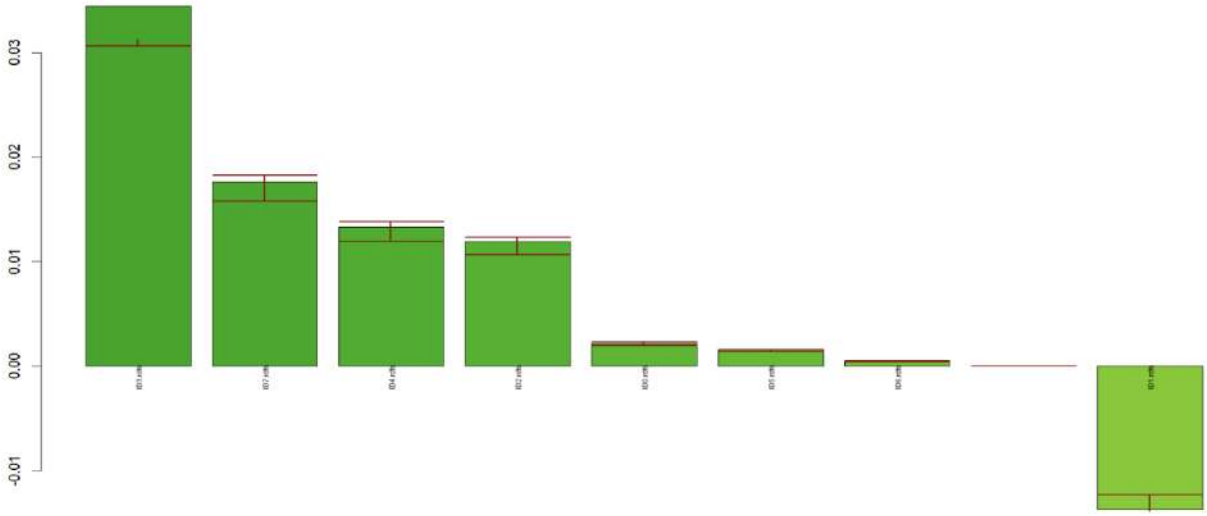
Malheureusement, en arrivant sur la fin du stage, il me sera difficile de les mettre en place de manière rapide et efficace. Ainsi, la mission de la création des groupes de règles a été déléguée à ChatGPT4o (dont voici la conversation **[63]**).

Dans cette conversation, je lui ai donné l'ensemble des sujets des triplets $< s, p, o >$ représentant les règles créées lors de la rédaction de l'article et après. Je lui ai ensuite demandé de me les classer en autant de clusters indépendants qu'il le souhaite (finalement les clusters n'étaient pas indépendants en sortie, mais ce n'était pas une nécessité fondamentale). Cela a abouti sur la création de huit clusters que nous décrivons dans le tableau ci-dessous :

Cluster	Description
Famille et ascendance	Contient l'ensemble des règles ayant attrait à la famille : parent, enfant, fratrie, ascendance, descendance, etc.
Relations et associations	Contient l'ensemble des règles permettant de décrire une affiliation : partis politiques, équipes sportives, influences, autorités, possesseurs, etc.
Localisations et géographie	Contient l'ensemble des règles décrivant un environnement, un statut géographique ou pointant vers un emplacement : capitales, pays, zones administratives, lieux de travail, affluents, etc.
Œuvres et professions créatives	Contient l'ensemble des règles décrivant les milieux culturels ou ce qui s'y associe : producteur, auteur, publications, inventeurs, architectes, etc.
Scientifique	Contient l'ensemble des règles portants sur les domaines scientifiques (principalement médical ou biologique) : études, taxonomies, médicaments, etc.
Temps et dates	Contient l'ensemble des règles traitant de la description temporelle des objets : date de mort, période historique, date d'ouverture officielle, etc.
Administratif et structurel	Contient l'ensemble des règles régissant les structures et administrations : corps législatif, maisons mères, succursales, etc.
Divers	Contient les autres règles : sous classe de, langue originale d'un film ou d'une série télévisée, etc.

Il suffit maintenant de répartir les règles correspondantes au sein de chaque groupe et d'utiliser la même méthode d'évaluation que tout à l'heure (usant de la dernière mesure).

Cela nous donne pour résultat, avec le RPL et la deuxième mesure, ceci :



Cela peut se traduire ainsi (résultat agrégé avec une analyse de l'ALG **[62]**) :

Cluster	Évolution de la corrélation en RPL
Famille et ascendance	**Hausse légère**
Relations et associations	**Baisse importante**
Localisations et géographie	**Importante hausse**
Œuvres et professions créatives	**Très importante hausse**
Scientifique	**Importante hausse**
Temps et dates	**Hausse légère** (baisse légère pour l'ALG)
Administratif et structurel	**Hausse légère**
Divers	**Importante hausse**

À partir de là, nous allons créer trois schémas RDFS/OWL à tester. Le premier avec les clusters résultant sur une hausse importante de la corrélation en RPL, le deuxième avec ceux disposant d'une hausse simple de leur corrélation en RPL et le troisième ceux disposant d'une hausse simple sur l'ALG et le RPL. Finalement, les seules relations qui ne seront jamais présentes sont celles du cluster « Relations et associations ».

Voici donc les résultats des nouvelles corrélations :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, article)	0.2691	0,2401	N/A
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes
RULK (KG, OntologiesV1 RDFS & OWL)	0,3107	**0,2903**	N/A ~1 heure
RULK (KG, AfterCorrelationGainAnalysis_RPL)	0,2991	0,2695	38 minutes
RULK (KG, Cluster Hausse RPL++)	0,3106	0,2781	53 minutes
RULK (KG, Cluster Hausse RPL+)	0,3111	0,2776	53 minutes
RULK (KG, Cluster Hausse RPL+\Temps et Dates)	**0,3136**	0,2799	54 minutes

Avec ces résultats, nous observons bien une hausse des corrélations en ALG, mais une baisse générale en RPL. Cela paraît surprenant dans la mesure où l'on avait à la base optimisé les résultats pour le RPL. Que l'ALG augmente, cela est cohérent, mais que le RPL baisse cela ne l'est pas.

Nous allons donc essayer une dernière fois avec une nouvelle ontologie partant du même principe, mais ayant pour règles dans chaque cluster, uniquement celles qui sont interdépendantes, c'est-à-dire celles qui permettent d'inférer une relation qui sera ensuite inférée par une autre règle par exemple.

Celle-ci est constituée de 45 clusters différents et voici les résultats finaux (avec la même démarche d'évaluation que précédemment :

Modèle (seuil 80%)	ALG	RPL	Temps d'exécution
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes
RULK (KG, OntologiesAfterDependentClusteredAnalysis)	0,3112	0,2822	54 minutes

Ici, nous observons bien une amélioration de la corrélation en ALG et en RPL suffisante pour dépasser la référence envoyée avec l'article. Néanmoins, cela ne dépasse pas pour autant l'ontologie (OntologiesV1) faite à la main dans la corrélation en RPL.

Nous observons donc bien plusieurs choses avec ces clusters :

1. **L'analyse des règles semble mieux une fois que celles-ci sont regroupées**, cela est plutôt logique dans la mesure où une règle seule n'aura pas autant d'impact qu'une série de règles s'utilisant mutuellement.
2. **L'évaluation d'une ontologie de manière quantitative n'est pas forcément la meilleure option.** En effet, celle-ci nous donne parfois des résultats que l'on a du mal à comprendre (augmentation de la corrélation en ALG et diminution de celle en RPL par exemple), ce qui nous rassure d'un certain côté quant au fait que les créer à la main de manière réfléchie semble être plus pertinent. Cf. « Ingénierie des connaissances ».

Néanmoins, ces résultats ne sont pas une excuse pour ne pas investiguer et comprendre le comportement de l'ontologie précédente. Cette investigation a malheureusement découlé sur une découverte assez embêtante expliquée plus en détail dans la partie **Erratum**.

Erratum : Le bug de la réflexivité

Avant de commencer cette partie, sachez que la problématique soulevée est présente à la toute fin de la partie **Bonus**.

Comme vous l'avez peut-être remarqué, ce rapport de stage s'est fait au fil de l'eau, j'écris donc cette partie 4 jours avant la remise du rapport. Néanmoins celle-ci reste essentielle, car elle met en lumière une erreur importante ayant été faite dans ce stage et les solutions qui y ont été apportées.

IMPORTANT : La résolution de ce problème est à date de rendu du rapport toujours en cours, les résultats et solutions présentés ici ne sont donc pas définitifs.

Brièvement, l'erreur a été soulevée en regardant les résultats des corrélations en RPL et en ALG dans la partie **Bonus**, ceux-ci indiquaient que si la corrélation en ALG augmentait, celle en RPL diminuait. En réalité cela vient d'un problème qui intègre dans le graphe de l'utilisateur, entre autres, des relations **réflexives [27]**. Cela s'introduit a priori quand la **transitivité** est utilisée simultanément avec **l'inverse ou la symétrie** (l'étendue du bug n'est pas entièrement connue, en sachant qu'il agit également quand RDFS est seul).

Ainsi, si « le frère de mon frère est mon frère » et que la fratrie contient deux individus, alors je suis mon propre frère, car la relation est à la fois transitive et symétrique. Mais pourquoi cela pose problème ?

Pour commencer, bien que nous cherchions les connaissances implicites, la réflexivité n'apporte pas de réelle valeur ajoutée dans le cadre RULK, au contraire, cela fait augmenter le temps de calcul tout en augmentant artificiellement le nombre de correspondances entre les triplets du graphe de l'utilisateur et de la cible. Il y a donc **surestimation de la corrélation.** Et cela impacte **TOUS** les résultats qui ont été effectués avec OWL ou RDFS, que ce soit avant ou après l'envoi de l'article.

Pour régler le problème tout en assurant la rétrocompatibilité des anciens résultats dans le code, j'ai ajouté un paramètre à la fonction du calcul de la similarité permettant de supprimer dans le graphe l'ensemble des triplets : $< s, p, o > tel\ que\ s = o$.

Je vais donc à partir de là, dresser le tableau des modifications des résultats les plus importants (non définitifs) :

Modèle (seuil 80%)	ALG (Réflexif)	RPL (Réflexif)	Temps d'exécution (Réflexif)	ALG (Non réflexif)	RPL (Non réflexif)	Temps d'exécution (Non Réflexif)
RULK (KG, article)	0.2651	0,2362	N/A	0.2651	0,2362	N/A
RULK (KG, PatchLocationV0 RDFS & OWL)	0,3063	0,2802	50 minutes	0,2725	0,2490	34 minutes
RULK (KG, PatchLocationV0 RDFS)	0,3015	0,2689	35 minutes	0,2773	0,2458	24 minutes
RULK (KG, PatchLocationV0 OWL)	0,2983	0,2696	49 minutes	0,2717	0,2471	32 minutes
RULK (KG, RDFS & OWL, référence)	0,3063	0,2802	50 minutes	0.2725	0.2490	50 minutes
RULK (KG, OntologiesV1 RDFS & OWL)	0,3107	**0,2903**	N/A ~1 heure	0,2729	0,2545	37 minutes
RULK (KG, AfterCorrelationGainAnalysis_RPL)	0,2991	0,2695	38 minutes	0,3041	0,2757	31 minutes
RULK (KG, Cluster Hausse RPL++)	0,3106	0,2781	53 minutes	0,3001	0,2694	33 minutes
RULK (KG, Cluster Hausse RPL+)	0,3111	0,2776	53 minutes	0,3059	0,2758	37 minutes
RULK (KG, Cluster Hausse RPL+\Temps et Dates)	**0,3136**	0,2799	54 minutes	0,2989	0,2694	35 minutes
RULK (KG, OntologiesAfterDependentClusteredAnalysis)	0,3112	0,2822	54 minutes	**0,3061**	**0,2764**	36 minutes

Globalement, nous observons des baisses importantes des résultats par rapport à leurs versions réflexives. Ainsi, nous observons une baisse moyenne de la corrélation en ALG de -9,33% et de -9,36% en RPL pour les résultats présents dans l'article, bien qu'ils restent en augmentation moyenne de +3,29% en ALG et +4.70% en RPL par rapport à la Baseline.

Nous observons également une baisse des résultats sur nos ontologies de manière générale, bien que celles-ci soient moins prononcées.

Finalement, nous avons donc deux bonnes nouvelles dans notre malheur : une meilleure cohérence des résultats et un temps de calcul réduit d'un facteur de presque 1,5.

Concernant les résultats pour les combinaisons de modèles, voici les corrigés pour ceux de quatre modèles :

Modèle (seuil 80%)	ALG (Réflexif)	RPL (Réflexif)	ALG (Non réflexif)	RPL (Non réflexif)
RULK (KG, KW, NE, LM, article)	0,36044	0,36163	0,36044	0,36163
RULK (KG OWL, KW, NE, LM) [PatchLocationV0]	0,37127	0,36960	0,37069	0.36983
RULK (KG RDFS, KW, NE, LM) [PatchLocationV0]	**0,37571**	0,37165	**0,37553**	**0,37142**
RULK (KG RDFS & OWL, KW, NE, LM) [PatchLocationV0]	0,37537	**0,37401**	0,36906	0,36884
RULK (KG, KW, NE, LM, article)	0,36044	0,36163	0,36044	0,36163
RULK (KG RDFS & OWL, KW, NE, LM) [AfterDependentClusteredAnalysis]	**0,38030**	**0,37696**	**0,38563**	**0,37994**

Ici, nous observons une variation plutôt en faveur du négatif dans les corrélations avec la nouvelle version en RPL et en ALG. Néanmoins, cette baisse reste modérée en valeur absolue, voire non conséquente pour OWL par exemple. Cependant, nous observons que RDFS engendre de meilleurs résultats que OWL ou RDFS & OWL, cela est plutôt perturbant dans la mesure où les deux sont censés représenter des aspects différents de la connaissance. Cela reste donc en cours d'investigation.

Il est tout de même intéressant de noter que nous obtenons aussi des résultats supérieurs pour la combinaison de quatre modèles avec la mise en place de clusters de

règles interdépendantes, que ce soit pour la corrélation en ALG, mais aussi en RPL. Ceci est une bonne nouvelle, car cela veut dire que l'on a réussi à dépasser en performances celles engendrées par le bug.

Maintenant, **que va-t-il arriver à l'article envoyé à CIKM2024 ?** Eh bien, cela dépendra de si l'article est accepté ou non. Si l'article est accepté, alors nous le corrigerons avant de le renvoyer en Camera-Ready le 08 août (23h59 AoE), si ce n'est pas le cas, alors nous pourrons dans un nouvel article pour une autre conférence l'améliorer en corrigeant les résultats et en incluant de nouvelles composantes telles que l'évaluation des ontologies par exemple.

Je resterai donc en contact avec mes maîtres de stage à minima à moyen terme, afin de répondre aux questions si besoin et de leur transmettre l'ensemble des résultats corrigés.

VI – Conclusion, difficultés, orientation et remerciement

Pour conclure, ce stage m'aura permis, outre de confirmer mon attirance pour le monde de la recherche, d'apprendre ou d'approfondir de nombreuses choses parmi lesquelles :

- **Les graphes**, pour lesquels je ne possédais aucune expérience que ce soit en mathématiques ou en informatique. Cela m'a permis de comprendre qu'au-delà d'un beau dessin, ceux-ci peuvent avoir des applications bien concrètes dans la recherche. En dehors du stage, je me suis également intéressé à leur application, au niveau personnel, en informatique avec les bases de données sous forme de graphes, mais aussi en mathématiques avec les chaînes de Markov que je trouve assez intéressantes.

- **Les environnements virtuels**, où là aussi je n'avais quasiment pas de connaissances sur ces derniers, mais surtout à quoi pouvaient-ils servir. Maintenant, je comprends leur utilité dans le cadre de projets, pour ne pas mélanger les versions des modules ou du langage de programmation par exemple.

- **Le web sémantique**, où jusque-là, son existence m'était simplement inconnue, alors que ça englobe des concepts et outils extrêmement intéressants tels que les données liées, la sémantique, RDF, RDFS/OWL, SPARQL, DBpedia ou même la logique de description, des prédicats et bien d'autres qui pourraient être grandement utiles notamment dans la formalisation des règles pour les IA symboliques.

Au niveau des difficultés rencontrées lors de ce stage, il n'y en a pas eu d'insurmontable d'un point de vue technique. Néanmoins, certaines ont tout de même fait perdre un peu de temps :

- Les résultats de l'étude détaillant $RULK_{KG}$ n'étaient pas les mêmes que ceux reproduits par moi-même. Après analyse du code, des fichiers entrants et des versions des packages, nous n'avons pas trouvé d'erreurs. Celle-ci se trouvait en fait dans une version différente du modèle de langage, utilisé par un des modules que l'on utilisait. La solution a simplement été la mise en place d'un environnement virtuel et d'un fichier `requirement.txt` complet. Soit, en réinstallant toutes les libraires présentes dans le fichier.

- Le temps pharaonique du calcul de la corrélation avant le passage à RDF qui prenait l'entièreté de la journée de stage, sans que je puisse faire quoi que ce soit d'autre de concret sur le code pendant l'exécution. La solution aura donc été le passage à RDF, mais avant ça de lancer les calculs en arrivant chez soi le soir et de les répertorier le lendemain matin en se levant.

- Une documentation complète pour RDF, OWL et RDFS, mais assez dure à déchiffrer au début. De plus, pour la mise en place d'ontologies, nous trouvions au début principalement des aides pour en créer sur Protégé **[70]**, un logiciel de gestion d'ontologies, mais pas directement dans un schéma RDFS/OWL comme je l'ai fait. Cela noie un peu plus les bonnes informations/documentations, quand celles sur les ontologies sont déjà bien assez dures à trouver.

Finalement, dans ce stage nous aurons découvert RULK, un cadre permettant l'estimation du gain de connaissance d'une personne dans le contexte de la recherche en tant qu'apprentissage. Dans ce contexte, où l'estimation du gain de connaissance est essentielle pour mettre en place des systèmes de recherche d'informations plus performants, nous observons qu'une partie des connaissances passent sous silence dans nos représentations, car elles ne sont pas explicitées. Ce stage aura permis de faire un premier pas vers la prise en compte de ces connaissances implicites pour avoir une représentation plus complète des connaissances de l'utilisateur, tout en permettant d'ouvrir une piste sur de futures améliorations.

Ainsi, j'ai pu participer dans le cadre de ce stage à un processus de recherche quasi-complet, observer et parler avec plusieurs chercheurs et doctorants en thèse classique ou CIFRE, qui ont pu me décrire ce qu'ils faisaient et comment se déroulait un doctorat. De plus, le mode de travail tel que je l'ai eu lors de ce stage me conviendrait parfaitement dans le cadre de ma future vie professionnelle. C'est par tout cela que ce stage m'a bel et bien conforté dans mon optique de continuer dans le secteur de la recherche en informatique.

Concernant mon orientation, je pars l'année prochaine pour accomplir ma troisième année de BUT SD en échange Erasmus, en Autriche, à l'université JKU de Linz **[64]**. Là-bas, je continuerai la science des données en anglais, mais avec un accent sur l'IA symbolique et connexionniste, en plus d'une ouverture sur les systèmes robotiques et flous qui je l'espère me permettront de pouvoir aborder des process plus particuliers si je le souhaite à l'avenir. Pour le Master, j'en cherche déjà un dans le domaine de l'IA, mais ouvert sur l'informatique et la science des données. Le master MINDS (IA) à l'INSA Lyon **[65]** pourrait être un exemple de ce que je recherche, mais je ne suis pas fixé. Ce qui est sûr, est que ma zone de recherche s'étend à l'UE entière, pour des cursus en anglais, en français ou éventuellement en allemand en fonction du niveau dont je disposerai à la fin de l'échange en Autriche.

Je remercie sincèrement Mme. Pereira sans qui je n'aurais pas pu faire ce stage, mais aussi M. Tettamanzi pour m'avoir aussi bien conseillé lors des réunions hebdomadaires et M. Nasser pour m'avoir aidé à résoudre les problèmes quand j'en avais.

Plus généralement, je remercie également M. Soliveres de m'avoir accompagné et de m'avoir supporté ces deux dernières années dans les longues démarches pour partir en Autriche.

Annexes, sources et documentation

Dans cette partie, vous trouverez l'ensemble des documents / sites utilisés pour mener à bien les missions du stage. Concernant le développement technique en lui-même, vous pourrez le retrouver sur GitHub pour les versions sans RDF et celle avec RDF[17]. La documentation et ces développements (pour RDF) pourront également se trouver dans les annexes, avec l'ensemble du projet **[71]**.

Note : Une partie importante des sources et des annexes sont en Anglais, dans ce cas, le titre de l'annexe/de la source sera succédée de la pastille **(en)**.

[00] Convention de stage

Lien : Document en annexe

[01] Informations complémentaires sur ELIZA

Source : Wikipedia
Lien : https://fr.wikipedia.org/wiki/ELIZA
Auteurs : https://fr.wikipedia.org/w/index.php?title=ELIZA&action=history
Date de publication : 1er février 2003
Dernière mise à jour à date d'écriture : 7 mars 2024

[02] GitHub ELIZA avec plongements lexicaux (en)

Source : GitHub
Lien : https://github.com/LesageArno/ELIZA_WordEmbedding
Auteurs : Arno LESAGE (basé sur le travail de Wade Brainerd)
Date de publication : Juillet 2023
Dernière mise à jour à date d'écriture : 21 janvier 2024

[03] RULK: A Framework for Representing User Knowledge in Search-as-Learning (en)

Source : Papier (Desires 2022)
Lien : https://ceur-ws.org/Vol-3480/paper-01.pdf
Auteurs : Arthur Câmara, Dima El Zein, Célia da Costa Pereira
Date de publication : N/A

[04] $RULK_{KG}$: Estimating User's Knowledge Gain in Search-as-Learning Using Knowledge Graph (en)

Source : Papier (à paraitre dans CHIIR 2024)
Lien : https://inria.hal.science/hal-04567545/file/Association_for_Computing_Machinery__ACM____SIG_Proceedings_Template__2_.pdf
Auteurs : Hadi Nasser, Dima El Zein, Célia Da Costa Peireira, Cathy Escazut, Andrea Tettamanzi
Date de publication : N/A

[17] Si l'accès vous ai refusé et que vous voulez voir le projet sur GitHub, n'hésitez pas à m'envoyer un message par mail.

[05] Informations complémentaires sur BERT (en)

Source : Wikipédia
Lien : https://en.wikipedia.org/wiki/BERT_(language_model)
Auteurs : https://en.wikipedia.org/w/index.php?title=BERT_(language_model)&action=history
Date de publication : 7 septembre 2020
Dernière mise à jour à date d'écriture : 5 mars 2024

[06] $RULK_{NE}$: Representing User Knowledge State in Search-As-Learning with Named Entities (en)

Source : Papier (CHIIR 2023)
Lien : Document en annexe
Auteurs : Dima El Zein, Arthur Câmara, Célia Da Costa Pereira, Andrea Tettamanzi
Date de publication : N/A

[07] Sujet Stage I3S

Lien : Document en annexe

[08] Site officiel du laboratoire I3S

Lien : https://www.i3s.unice.fr/fr/

[09] Site officiel du CNRS

Lien : https://www.cnrs.fr/fr

[10] Site officiel de l'INS2I

Lien : https://www.ins2i.cnrs.fr/fr

[11] Site officiel de l'Université côte d'azur

Lien : https://univ-cotedazur.fr/

[12] Site officiel de l'INRIA (côte d'Azur)

Lien : https://www.inria.fr/fr/centre-inria-universite-cote-azur

[13] Rapport d'activité 2022 CNRS

Source : CNRS
Lien : https://www.cnrs.fr/fr/le-cnrs/rapport-d-activite
Auteurs : Sophie Felix, Laurence Stenvot, Dina Tiouti
Date de publication : N/A

[14] Budget initiale de l'Université Côte d'Azur en 2022

Source : Université Côte d'Azur
Lien : https://univ-cotedazur.fr/medias/fichier/2021-138-budget-initial-2022_1643641552215-pdf
Auteurs : Marc Dalloz
Date de publication : 20 décembre 2021

[15] Publications du laboratoire I3S

Source : HAL
Lien : https://cnrs.hal.science/I3S
Auteurs : dépend des articles
Date de publication : dépend des articles

[16] Logiciels développés par le laboratoire I3S

Source : HAL
Lien : https://cnrs.hal.science/I3S/search/index/?q=docType_s:SOFTWARE
Auteurs : dépend des logiciels
Date de publication : dépend des logiciels

[17] Informations complémentaires sur les CDO (en)

Source : Wikipédia
Lien : https://en.wikipedia.org/wiki/Collateralized_debt_obligation
Auteurs : https://en.wikipedia.org/w/index.php?title=Collateralized_debt_obligation&action=history
Date de publication : 22 décembre 2004
Dernière mise à jour à date d'écriture : 26 mars 2024

[18] GitHub de REBEL (en)

Source : GitHub
Lien : https://github.com/Babelscape/rebel
Auteurs : Pere-Lluís Huguet Cabot, Roberto Navigli, Babelscape
Date de publication : 22 octobre 2021
Dernière mise à jour à date d'écriture : 9 novembre 2023

[19] Site officiel de DBpedia (en)

Lien : https://www.dbpedia.org/

[20] Informations complémentaires sur les ontologies en informatique

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Ontologie_(informatique)
Auteurs : https://fr.wikipedia.org/w/index.php?title=Ontologie_(informatique)&action=history
Date de publication : 26 octobre 2012
Dernière mise à jour à date d'écriture : 12 avril 2024

[21] Pere-Lluís Huguet Cabot, Roberto Navigli. REBEL: Relation Extraction By End-to-end Language generation (en)

*Source : **[18]***
*Lien : Document en annexe, **[18]***

[22] Informations complémentaires sur le module Python nltk (en)

Source : PyPi
Lien : https://pypi.org/project/nltk/
Auteurs : https://pypi.org/project/nltk/
Date de publication (+ version) : 3 juillet 2007 (0.8)
Dernière mise à jour à date d'écriture (+ version) : 2 janvier 2023 (3.8.1)
Installation : pip install nltk

[23] Informations complémentaires sur la Tokenisation (en)

Source : Wikipédia
Lien : https://en.wikipedia.org/wiki/Lexical_analysis#Tokenization
Auteurs : https://en.wikipedia.org/w/index.php?title=Lexical_analysis&action=history
Date de publication : 27 février 2020
Dernière mise à jour à date d'écriture : 17 avril 2024

[24] Homonymes de CDO sur Wikipédia

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/CDO
Auteurs : https://fr.wikipedia.org/w/index.php?title=CDO&action=history
Date de publication : 14 octobre 2007
Dernière mise à jour à date d'écriture : 15 juillet 2021

[25] Différence espèces endémiques et espèces indigènes

Source : ProjetEcolo
Lien : https://www.projetecolo.com/espece-endemique-definition-et-exemples-1276.html
Auteurs : « Équipe éditoriale »
Date de publication : 11 avril 2023

[26] Notice pour l'utilisation de sentence-transformers/all-MiniLM-L6-v2 (en)

Source : Hugging Face
Lien : https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
Auteurs : https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2/commits/main
Date de publication : 30 août 2021
Dernière mise à jour à date d'écriture : 27 mars 2024

[27] Informations complémentaires sur les liens réflexifs

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Relation_r%C3%A9flexive
Auteurs : https://fr.wikipedia.org/w/index.php?title=Relation_r%C3%A9flexive&action=history
Date de publication : 4 octobre 2006
Dernière mise à jour à date d'écriture : 7 juin 2022

[28] Informations complémentaires sur la définition de la transitivité

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Relation_transitive
Auteurs : https://fr.wikipedia.org/w/index.php?title=Relation_transitive&action=history
Date de publication : 10 mars 2005
Dernière mise à jour à date d'écriture : 25 avril 2023

[29] The Church-Turing Thesis (en)

Source : The Stanford Encyclopedia of Philosophy
Lien : https://plato.stanford.edu/archives/spr2024/entries/church-turing/
Auteurs : Copeland, B. Jack
Date de publication : 8 janvier 1997
Dernière mise à jour à date d'écriture : 18 décembre 2023

[30] Can every recursion be converted into iteration? (en)

Source : Stack Overflow
Lien : https://stackoverflow.com/questions/931762/can-every-recursion-be-converted-into-iteration
Auteurs : N/A
Date de publication : Juin 2009
Dernière mise à jour à date d'écriture : Août 2021

[31] SearchX: Empowering Collaborative Search Research (en)

Source : Association for Computing Machinery
Lien : https://dl.acm.org/doi/10.1145/3209978.3210163
Auteurs : Rikarno Putra, Felipe Moraes, Claudia Hauff
Date de publication : Juin 2018

[32] Site officiel de SpaCy (en)

Lien : https://spacy.io/

[34] Résultat améliorations.xlsx

Lien : Document en annexe
Auteurs : Arno Lesage

[35] Cours INRIA : Web sémantique et Web de données

Source : France Université Numérique (FUN)
Lien : https://www.fun-mooc.fr/fr/cours/web-semantique-et-web-de-donnees/
Auteurs : Fabien Gandon, Olivier Corby, Catherine Faron Zucker (INRIA)
Date de diffusion : 4 février 2019 – 31 décembre 2024

[36] Informations complémentaires sur le web sémantique

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Web_s%C3%A9mantique
Auteurs : https://fr.wikipedia.org/w/index.php?title=Web_s%C3%A9mantique&action=history
Date de publication : 3 décembre 2018
Dernière mise à jour à date d'écriture : 20 avril 2024

[37] Informations complémentaires sur les URI et les IRI (en)

Source : World Wide Web Consortium (W3C)
Lien : https://www.w3.org/Addressing/
Auteurs : https://www.w3.org/Addressing/#time
Date de publication : 1991
Dernière mise à jour à date d'écriture : Janvier 2005

[38] Informations complémentaires sur XML 5 (en)

Source : World Wide Web Consortium (W3C)
Lien : https://www.w3.org/TR/2008/REC-xml-20081126/
Auteurs : Tim Bray, Jean Paoli, Sperberg-McQueen, Eve Maler, François Yergeau
Date de recommandation : 26 novembre 2008

[39] Informations complémentaires sur RDF (en)

Source : World Wide Web Consortium (W3C)
Lien : https://www.w3.org/TR/rdf-concepts/
Auteurs : Graham Klyne, Jeremy J. Carroll, Brian McBride
Date de recommandation : 25 février 2014

[40] Site officiel du W3C (en)

Lien : https://www.w3.org/

[41, 42, 43] Informations complémentaires sur les syntaxes de RDF (en)

Source : World Wide Web Consortium (W3C)
*Lien **[41]** : https://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/*
*Auteurs **[41]** : Fabien Gandon, Guus Schreiber*
*Lien **[42]** : https://www.w3.org/TR/2014/REC-turtle-20140225/*
*Auteurs **[42]** : David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, Gavin Carothers*
*Lien **[43]** : https://www.w3.org/TR/n-triples/*
*Auteurs **[43]** : Gavin Carothers, Andy Seaborne, David Beckett*
Date de recommandation : 25 février 2014

[44] Informations complémentaires sur le vocabulaire FOAF (en)

Source : xmlns
Lien : http://xmlns.com/foaf/spec/
Auteurs : http://xmlns.com/foaf/spec/
Date de publication : 24 mai 2007
Dernière mise à jour à date d'écriture : 14 janvier 2014

[45] Informations complémentaires sur les moteurs d'inférences OWL

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Web_Ontology_Language#Moteurs_d'inf%C3%A9rences
Auteurs : https://fr.wikipedia.org/w/index.php?title=Web_Ontology_Language&action=history
Date de publication : 29 octobre 2004
Dernière mise à jour à date d'écriture : 4 mars 2024

[46] Informations complémentaires sur les profils de OWL (en)

Source : World Wide Web Consortium (W3C)
Lien : https://www.w3.org/TR/owl2-profiles/
Auteurs : Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, Diego Calvanese, Jeremy Carroll, Giuseppe De Giacomo, Jim Hendler, Ivan Herman, Bijan Parsia, Peter F. Patel Schneider, Alan Ruttenberg, Uli Sattler, Michael Schneider
Date de recommandation : 11 décembre 2012

[47] Page du logiciel Corese (en)

Lien : https://project.inria.fr/corese/

[48] Github de Corese (documentation Corese Python) (en)

Source : GitHub
Lien : https://github.com/Wimmics/corese/blob/master/docs/corese-python/Corese-library%20with%20Python.md
Auteurs : https://github.com/Wimmics/corese/
Date de publication : 14 mars 2017
Dernière mise à jour à date d'écriture : 13 mars 2024

[49] Site officiel de Py4J (en)

Lien : https://www.py4j.org/

[50] RDFLib (7.0.0) (en)

Source : Zenodo
Lien : https://zenodo.org/records/8206632
Auteurs : https://zenodo.org/records/8206632
Date de publication : 16 juillet 2022
Dernière mise à jour à date d'écriture : 1er août 2023
Installation : pip install rdflib

[51] OWL-RL: OWL-RL: A simple OWL2 RL reasoner on top of RDFLib (en)

Source : Zenodo
Lien : https://zenodo.org/records/14543
Auteur : Ivan Herman
Date de publication : 14 octobre 2014
Installation : pip install owlrl

[52] Topic-independent modeling of user knowledge in informational search sessions (en)

Source : Springer Link
Lien : https://link.springer.com/article/10.1007/s10791-021-09391-7
Auteurs : Ran Yu, Rui Tang, Markus Rokicki, Ujwal Gadiraju et Stefan Dietze
Date de publication : 16 mars 2021

[53] Informations complémentaires sur la logique floue

Source : Wikipédia
Lien : https://fr.wikipedia.org/wiki/Logique_floue
Auteurs : https://fr.wikipedia.org/w/index.php?title=Logique_floue&action=history
Date de publication : 2 octobre 2003
Dernière mise à jour à date d'écriture : 26 décembre 2023

[54] Site officiel de CIKM 2024 (en)

Lien : https://cikm2024.org/

[55] Site officiel de Overleaf

Lien : https://fr.overleaf.com/

[56] Enhancing User's Knowledge Gain Estimation in Search-as-Learning Using Implicit Knowledge (en)

Source : Stage
Lien : Document en annexe
Auteurs : Arno Lesage, Hadi Nasser, Célia Da Costa Pereira, Cathy Escazut, Andréa Tettamanzi
Date de publication : N/A

[57] Définition de base et étymologie du mot « ontologie »

Source : La langue française
Lien : https://www.lalanguefrancaise.com/dictionnaire/definition/ontologie
Dernière mise à jour à date d'écriture : 8 avril 2024

[58] A translation approach to portable ontology specifications (en)

Source : Science Direct
Lien : https://www.sciencedirect.com/science/article/pii/S1042814383710083?via%3Dihub
Auteurs : Thomas R. Gruber
Date de publication : Juin 1993

[59] Ontologies du Web : Histoire refoulée et perspectives paradoxales

Source : Persée
Lien : https://www.persee.fr/doc/intel_0769-4113_2014_num_61_1_1041
Auteurs : Aurélien Bénel
Date de publication : Janvier 2014

[60] OntologyRepository.xlsx (en)

Lien : Document en annexe
Auteurs : Arno Lesage

[61] Informations complémentaires sur le coefficient de corrélation de Pearson (en)

Source : Wikipédia
Lien : https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
Auteurs : https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&action=history
Date de publication : 5 mai 2003
Dernière mise à jour à date d'écriture : 2 juin 2024

[62] Analyse des règles des ontologies (RMD – PDF) (en)

Lien : Document en annexe
Auteurs : Arno Lesage

[63] Discussion sur la création des groupes de règles par ChatGPT4o (en)

Lien : Document en annexe
Auteurs : Arno Lesage

[64] Informations complémentaires sur l'université JKU de Linz (en)

Lien : https://www.jku.at/en

[65] Informations complémentaires sur le Master MINDS à l'INSA Lyon (en)

Lien : https://www.insa-lyon.fr/fr/formation/master-minds-ia

[67] Informations complémentaires sur les UMR

Source : CNRS
Lien : https://www.cnrs.fr/comitenational/cs/recommandations/10 - 11_octobre_2011/l'Unite_mixte_de_recherche_(UMR).pdf
Auteurs : Bruno Chaudret
Date de recommandation : 11 octobre 2011

[68] Informations complémentaires sur le module NetworkX (en)

Source : PyPi
Lien : https://pypi.org/project/networkx/
Auteurs : https://pypi.org/project/networkx
Date de publication (+ version) : 5 août 2005 (0.23)
Dernière mise à jour à date d'écriture (+ version) : 6 avril 2024 (3.3)
Installation : pip install networkx

[69] Informations complémentaires sur PIP (en)

Source : PyPi
Lien : https://pypi.org/project/pip/
Auteurs : https://pypi.org/project/pip/
Date de publication (+ version) : 28 octobre 2008 (0.2)
Dernière mise à jour à date d'écriture (+ version) : 21 juin 2024 (24.1)
Installation : pip install pip

[70] Informations complémentaires sur Protégé (en)

Source : Protégé – Stanford University
Lien : https://protege.stanford.edu/about.php
Auteurs : Mark A. Mussen, Centre de recherche pour l'informatique biomédicale du département de médecine de l'université de Stanford
Date de publication (+ version) : 11 novembre 1999 (1.1)
Dernière mise à jour à date d'écriture (+ version) : 30 mai 2024 (5.6.4)

[71] Code de l'ensemble du projet (en)

Lien : Document en annexe
Auteurs : Arno Lesage

[72] RDF 1.2 Schema (en)

Source : W3C
Lien : https://www.w3.org/TR/rdf12-schema/
Auteurs : Dominik Tomaszuk, Timothée Haudebourg, Guus Schreiber, Dan Brickley, R.V. Guha, Brian McBride
Date de publication : 16 mai 2023
Dernière mise à jour à date d'écriture : 18 avril 2024

[73] OWL 2 Web Ontology Language Document Overview (Second Edition) (en)

Source : W3C
Lien : https://www.w3.org/TR/owl-ref/
Auteurs : W3C OWL Working Group
Date de recommandation : 11 décembre 2012

[74] SPARQL 1.1 Query Language (en)

Source : W3C
Lien : https://www.w3.org/TR/sparql11-query/
Auteurs : Steve Harris, Andy Seaborne, Eric Prud'hommeaux
Date de recommandation : 21 mars 2013