الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
كلية العلوم الدقيقة و التطبيقية
Faculté des sciences exactes et appliquées
قسم الإعلام الآلي
Computer Science Department

Mémoire de Fin d'Etudes

Pour l'Obtention du Diplôme de
Master en Informatique

Présenté par :
HAMMOUDA FOUZI MOHAMMED AMINE

Domaine : Mathématiques & Informatique
Spécialité : *Système d'Information et Technologie du web*

Session 1 **2022**

THEME

ALIGNFX : PLATEFORME DE CORRESPONDANCES SÉMANTIQUES BASÉES SUR LE CALCUL DE PROBABILITÉ DANS L'ALIGNEMENT DES ONTOLOGIES

Encadré par : Mme MEZIANE Hassina
Co-encadré par : Mme EL GHANDOUR Naima

Jury

Président : Mme NAIT BAHLOUL Safia
Examinateur : Mr ABDI Mustapha Kamel

Code Master : 14/2022

Promotion 2021/2022

Abstract

Les ontologies sont un moyen de représentation et de stockage de l'information, elles aident à résoudre le problème de l'interopérabilité sémantique dans le web sémantique. Mais plus le temps passe plus les sources d'information prennent de l'ampleur provoquant ainsi l'émergence des hétérogénéités causées par les divergences conceptuelles entre les développeurs, c'est là où intervient l'alignement sémantique des ontologies.
L'alignement sémantique est l'opération qui prend en entré deux ou plusieurs ontologies et qui donne en sortie un alignement final représentant un ensemble de correspondances entre les concepts. Toutefois pour extraire les correspondances il est utilisé des méthodes de similarités/probabilités.

Mots-clés : *Alignement*, *correspondance*, *Aggregation*, *Similarité*, *Probabilité*.

Ontologies are a type of information representation and storage system. They help to solve the problem of semantic interoperability in the semantic web, but as time goes on, the size of the sources of information grows larger, and with it the heterogeneity that emerges as a result of conceptual divergences between developers. This is where semantic alignment of ontologies comes into play.
Semantic alignment is a process that takes two or more ontologies as input and produces a final alignment that is a set of correspondences between concepts. To extract the correspondences, However, methods for calculating similarities/probabilities must be used.

Keywords : *Alignment*, *Mapping*, *correspondence*, *Aggregation*, *Similarity*, *Probability*.

Remerciements

Je tiens à exprimer mes sincères remerciements aux professeurs, qui sont les piliers de SITW :

Tout d'abord ce projet est dédié à Mr BENAISSA M. qui a été l'un des professeurs le plus engagé dans l'éducation, ça gentillesse et son expérience m'ont été bénéfique, encore une fois merci beaucoup.

A Madame MEZIANE H., Maître de Conférence *A* à l'université d'Oran1, encadreur de ce mémoire. Un professeur qui a su me pousser dans mes limites, un professeur exceptionnel, avec qui j'ai pu m'instruire depuis ma deuxième année, qui se donne à fond pour ses étudiants, qui ma donné beaucoup de critiques constructives, sa disponibilité en tout temps et bien-sur sa patience avec moi lors de l'écriture de ce présent mémoire. Grâce à son expérience, j'ai pu apprendre beaucoup de choses et je vous suis reconnaissant pour ces années durant lesquelles vous m'avez appris énormément de choses, encore une fois merci.

A Madame NAIT BAHLOUL S., Professeur à l'université d'Oran1, présidente et examinatrice de ce mémoire, une enseignante qui avec beaucoup de gentillesse donne tout son possible pour faire parvenir ses connaissances et son expertise à ses étudiants qui est sûrement une de ses grandes qualité.
Qu'elle trouve ici l'expression de mes plus chaleureux et sincères remerciements.

Je tiens aussi à remercier Mr ABDI M.K. pour avoir accepté d'examiner mon travail et que je porte en grande estime.

J'adresse mes sincères remerciements à tous les professeurs, Mme. BARIGOU F., Mr ZEKRI L., Mme AMRANE F., Mme. MOKHTARI N., mon co-encadreur Mme. EL GHANDOUR N. et à toutes les personnes qui par leurs paroles, et leurs conseils ont su me guidé durant mon parcours et merci à eux d'avoir été à l'écoute.

Dédicaces

Du profond de mon coeur, je dédie ce travail à tout ceux qui me sont chers.

A Ma Chère Mère

Aucun mot ne saurait exprimer ma considération et mon amour pour toi, tu a sacrifié énormément pour mon éducation, seul dieu sait à quel point tu t'es donnée à fond pour mon éducation, mes bonnes manières, mon bien-être, et pour l'amour que tu me portes, je te remercie. Ton dévouement sera à jamais gravé dans m'a mémoire.

A Mon Cher Père

Quelque soit les mots que j'utilise, aucun ne peut exprimer le respect que j'ai pour toi, tu nous a quitté trop tôt, mais je sais que là ou tu es, tu veilles sur moi et j'essayerai toujours de te rendre fier de moi, de par mes études et par mes actions au quotidien.

A mes deux grand frères **Zoheir, Ilyes**, puisse dieu vous accorder un merveilleux avenir et une grande réussite.

Table des matières

Table des figures

Liste des Tableaux

Introduction Générale

Introduction Générale

Une ontologie fournit un moyen de décrire de l'information en le représentant avec un ensemble de termes et de concepts structuré, ou les concepts sont reliés entre eux par des relations sémantiques ou de subsomptions. La notion d'ontologie englobe plusieurs modèles de données, par exemple, des classifications, des schémas de bases de données etc... Elles ont tendance à être mises partout. Elle sont utilisées dans de nombreuses applications, telles que l'intégration d'information, les services Web sémantiques, les réseaux sociaux etc...Cependant le Web sémantique possède plusieurs sources d'information, qui prennent de l'ampleur, provoquant ainsi l'émergence des hétérogénéités causées par les divergences conceptuelles entre les développeurs, c'est la ou intervient **Alignement des ontologies**.

Alignement des ontologies est le processus qui détermine les correspondances entre concepts. Ces dernières peuvent être obtenues avec différentes méthodes . Ces méthodes se divisent en deux classes :

1. **Les méthodes de similarités** qui peuvent se basées sur la comparaison entre deux chaînes de caractères (respectivement entre deux concepts), ou sur les ensembles (respectivement l'intersection des instances des concepts).

2. **Les méthodes probabilistes** calculent la probabilité de subsomption de deux concepts en utilisant leurs instances.

Problématique

Dans l'alignement des ontologies, les méthodes de similarité (en particulier les mesures de distance) se concentrent sur la terminologie des concepts, ainsi, si ces concepts sont écrits différemment (fautes d'orthographes, langages naturels différents etc...), ces méthodes donnent des résultats erronés. En effet :

1. Nous pouvons avoir le cas où dans deux ontologies, qui décrivent un même domaine, il peut exister dans l'une d'elle des concepts enregistrés avec des fautes d'orthographe. Par exemple, book et buuk, ne peuvent pas être considérés comme similaires à cause des deux u dans le deuxième concept.

2. Les correspondances extraites par les méthodes de similarités sont des correspondances avec des relations d'équivalence, ce qui provoque une perte de ces dernières avec des relations d'inclusion ou de généralisation. Par exemple, Film et Art sont considérés comme similaires, car les films sont une forme d'art. Mais en utilisant les méthodes de similarités, ils seront définis comme non similaires.

Ainsi, la solution à ces problèmes consiste à utiliser des **méthodes de probabilité** car elles sont beaucoup plus efficaces et plus performantes pour l'extraction de l'alignement.

Objectifs

Les objectifs de notre travail consistent :

1. à prouver que l'alignement sémantique basé sur les méthodes de probabilités sont plus performantes que l'alignement basé sur les méthodes de similarités.
2. à concevoir et implémenter une plate-forme de correspondance sémantique basée sur le calcul de probabilité pour l'alignement des ontologies.

Contributions

Nos contributions dans ce travail, s'articulent autour des points suivants :

1. La création de nos ontologies à partir de deux taxonomies, *YAGO* et *DBpedia*.
2. Implémentation des algorithmes de similarité incluant :
   - *L'algorithme de JARO.*
   - *L'algorithme de Levenstein "Edit distance".*
   - *L'algorithme de JACARD.*
3. Implémentation des algorithmes de probabilité suivantes :
   - *L'algorithme de PARIS.*
   - *L'algorithme de ProbaMap.*
4. La comparaison entre tous les algorithmes cités ci-dessus, à travers le processus agrégation des correspondances pertinentes, afin de :
   - *Trouver le meilleur algorithme pour l'alignement des ontologies données.*
   - *Prouver que les algorithmes de probabilités sont plus performants et plus efficaces que les algorithmes de similarités.*

Structure du mémoire

Le présent travail est subdivisé en trois chapitres :

Le chapitre 1 est consacré à la définition de l'alignement des ontologies, suivi de quelques méthodes de similarités/ probabilités utilisées dans ce domaine.

Le chapitre 2 présente la partie conceptuelle de la solution proposée, le système d'alignement sémantique **AlignFX**, à travers une l'architecture globale du système.

Le chapitre 3 décrira l'implémentation du système AlignFX, ainsi que les résultats des expériences effectuées pour prouver que les méthodes probabilistes sont plus performantes que les méthodes de similarités.

Enfin, nous concluons notre travail par une récapitulation des solutions proposées et des perspectives futures pour la poursuite et l'amélioration de ce travail.

Etat de l'art

Chapitre 1

Etat de l'art

1.1 Introduction

Dans le monde actuel où le web sémantique est vu comme un système évolutif qui contient plusieurs ontologies, le besoin de combiner ces ontologies devient une nécessité, et la contrainte principale de cette opération est le fait que les données que contiennent ces ontologies sont hétérogènes, c'est là ou intervient **l'alignement des ontologies**. Plusieurs études, méthodes et articles ont vu le jour dans ce domaine, mais dans ce mémoire nous allons nous intéresser à la performance des méthodes probabilistes par rapport aux méthodes traditionnelles de similarité.

1.2 Web Sémantique et les Ontologies

1.2.1 Le Web Sémantique

Dans le monde d'aujourd'hui, le web est ouvert à tout le monde qui veut publier des informations, ces informations sont sous forme de page web ou de documents (HTML, WORD, PDF...) liés par des liens hypertextes, et sont lisibles et compréhensibles par l'humain seulement, car ils sont sous forme non-structurés et en langage naturel (Figure 1.1). Le stockage de ces informations en textes bruts a posé beaucoup de problèmes au niveau de l'intégration ou la réutilisation des données. Pour régler ces hétérogénéités et avoir une interopérabilité sémantique, il fallait ajouter **le Web Sémantique**.

Le Web sémantique vise à faciliter l'exploitation des données structurées, pour donner du sens au contenu des pages Web, en permettant leur interprétation par des machines. Il s'agit d'une extension du web actuel dans laquelle les ordinateurs ou les machines comprennent le sens des informations.

Les ressources du Web sémantique possèdent des identificateurs uniques, ce qui a permis de régler le problème d'hétérogénéité. Figure 1.1 illustre les langages utilisés dans le Web actuel et le Web sémantique. En effet, nous pouvons voir que dans le web actuel les langages utilisés sont sous forme de texte brut, HTML ou bien XML etc.. et ils sont utilisés pour la structuration des données. Alors que les langages du web sémantique, tels que RDF, RDFS et OWL, ils sont utilisés non seulement pour la structuration des données mais aussi pour annoter sémantiquement les ressources du web.

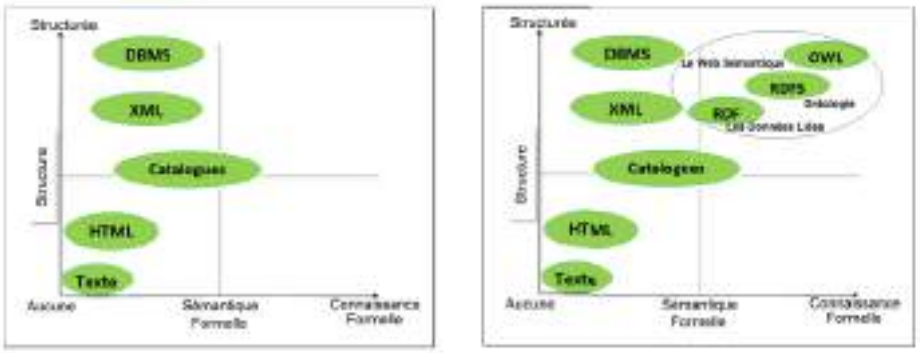


Figure 1.1: La structure du web actuel et du web sémantique [7]

1.3 Les Ontologies

1.3.1 Définition général

Dans l'informatique le terme ontologie est l'ensemble structuré de termes et de concepts représentant le sens d'un champ d'informations à travers une taxonomie des concepts du domaine, ainsi une ontologie est constituée d'un ensemble de concepts et de relations entre ces concepts.

Il existe plusieurs définitions d'ontologies :

Definition 1.3.1. Une ontologie est une spécification formelle explicite d'une conceptualisation partagée [5].

Definition 1.3.2. Une ontologie est une compréhension partagée d'un domaine d'intérêt [9].

1.3.2 Les composants d'une ontologie

Les composants d'une ontologie changent d'un langage à un autre, mais les composants les plus communs sont :

- **Les concepts ou classes** : ce sont un ensemble, une entité, une collection, ou un type d'objets, tous partageant des caractéristiques communes. Par exemple, le concept Personne est constitué de Mohammed, Fouzi, etc...Les relation entre classes construisent une taxonomie de classes.
- **Les instances** : ce sont les objets de base réels et concrets, par exemple, dans le concepts Personne, il y a l'instance Fouzi, Mohammed etc...
- **Les propriétés ou relations** : Les propriétés sont définies par le domaine, le co-domaine, les caractéristiques et les restrictions, elles peuvent normalement être exprimées directement entre individus ou entre concepts. On distingue deux types de propriétés :
  - **Les propriétés de type objet** : définissent les relations entre deux instances. Le domaine et le co-domaine des relations sont des classes. Ces mêmes relations peuvent êtres inversées par exemple, l'instance étudiant

étudie à l'université, son inverse serait, l'université contient l'étudiant. En plus de la propriété inverse, il existe d'autres caractéristiques pour les relations comme la transitive, la symétrique, l'antisymétrique, la réflexive et la fonctionnelle.

– **Les propriétés de type donnée** : définissent les relations entre instances et les valeurs de données. Les attributs sont des relations dans lesquelles le domaine est un concept, et le co-domaine est un type de données, comme par exemple, "String", "Integer", "Double", etc...

Les assertions peuvent être soit **une propriété de type données** qui spécifie des relations de type données sur une propriété donnée, soit **une propriété de type objet** qui spécifie des relations de type objet sur une propriété donnée.

- **Les restrictions** : une restriction permet de définir une classe par une contrainte qui porte sur les instances de la classe, cette contrainte s'exprime par une restriction sur le type des valeurs possibles d'une propriété.

- **Les axiomes** : Les axiomes sont la connaissance de base lors des raisonnements, elles peuvent êtres designées en RDFS/OWL, par exemple :

  Un homme est une personne peut ce traduire par ex:homme rdfs:subClassOf ex:personne, qui représente un seul axiome OWL-DL.

1.3.3 Exemple d'une ontologie

Figure 1.2 illustre un exemple d'ontologie [1], où les classes sont représentées comme des rectangles, les relations comme des hexagones et les instances comme des boites arrondies. Les relations de subsomption sont représentées comme des flèches continues et les flèches discontinues représentent des instanciations de classes et des relations. Il existe six classes : Object, Vehicle, Owner, Boat, Car, Speed; deux

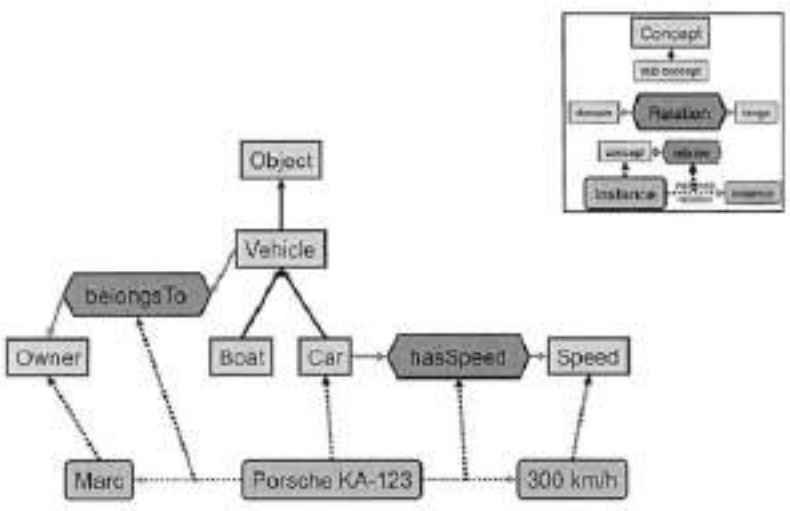


Figure 1.2: Exemple d'une ontologie [7]

relations : "belongsTo", "hasSpeed" et trois instances : Marc, Porsche KA-123, 300 km/h. Il existe une relations de subsomption entre Object et Vehicle, et entre Vehicle, Boat et car. Chaque voiture a une vitesse et un propriétaire, par exemple, la voiture Porsche KA-123 possède une vitesse de 300 km/h et appartient à Marc.

1.3.4 La Définition Formelle de l'Ontologie

Definition 1.3.3. [1] Une ontologie O est un six-uplet $<$ C, R, H^C , H^R, I, A $>$ où

- C : désigne l'ensemble des classes.
- R : désigne l'ensemble des relations.
- H^C : désigne une relation d'ordre partiel sur C appelée hiérarchie ou taxonomie des concepts. Elle associe à chaque concept ses sous-concepts.
- H^R : désigne une relation d'ordre partiel sur R appelée hiérarchie ou taxonomie des relations. Elle associe à chaque relation ses sous-relations.
- I : désigne l'ensemble des instances de C et R.
- A : désigne un ensemble d'axiomes.

1.4 L'alignement des ontologies

Les ontologies ont été adoptées pour régler le problème de l'interopérabilité sémantique, mais ces dernières ont seulement réduit l'hétérogénéité structurelle et syntaxique et donc ces ontologies sont remplis d'hétérogénéité à cause de différentes conceptions faites par l'humain. Pour assurer l'interopérabilité entre deux ou plusieurs ontologies **l'alignement des ontologies** a été proposée.

1.5 La Nécessité de l'Alignement des Ontologies

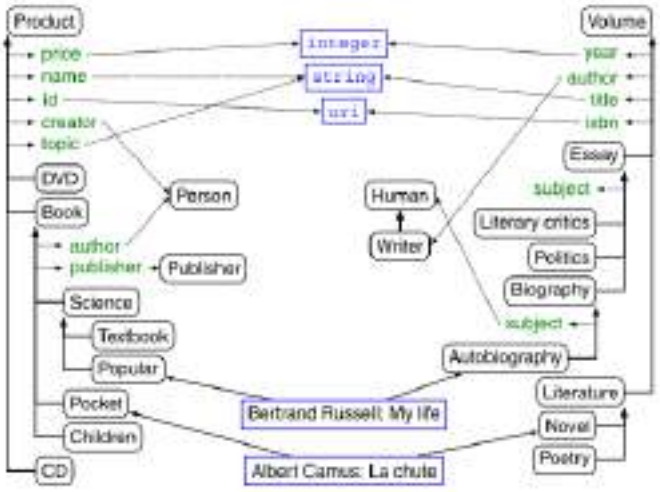


Figure 1.3: Exemple d'hétérogénéité d'ontologie [2]

L'ontologie est utilisée partout que ce soit pour décrire des éléments de la réalité ou stocker des connaissances, mais les ontologies sont hétérogènes entre elles et ces hétérogénéités mène a des erreurs lors des correspondances entre termes. Figure 1.3 illustre deux fragments de deux ontologies mettant en évidence leur hétérogénéité. La solution pour résoudre le problème de l'hétérogénéité est l'alignement des ontologies qui va permettre de combler le fossé sémantique entre les ontologies.

1.6 Les Différentes Formes d'hétérogénéité entre les Ontologies

Le rôle du l'alignement des ontologies est de réduire les hétérogénéité entre elles. Il existe plusieurs types d'hétérogénéité :

1.6.1 hétérogénéité au niveaux Syntaxique

Cette hétérogénéité survient quand les deux ontologies utilisent différents langages d'ontologie, par exemple, une ontologie utilisant RDF et une autre utilisant OWL, ou quand les deux ontologies utilisent différents formalismes de représentation des connaissances comme par exemple OWL et F-logic [2].

1.6.2 Hétérogénéité au niveau Terminologique

Cette hétérogénéité survient lors de l'utilisation de variation dont les noms faisant référence à la même entité dans différentes ontologies [2]. Les différentes causes peuvent être :

- L'utilisation de différents langages naturels (Anglais, Français, etc.) qui vont décrire le même concept, par exemple, "Apartment" en Anglais et "Appartement" en Français.
- L'utilisation de différents termes qui décrivent un même concept, par exemple, "maison" et "bercail".
- L'utilisation de l'homonymie, par exemple, "ver" et "verre".

1.6.3 Hétérogénéité au niveau Conceptuel

Cette hétérogénéité survient lors de la modélisation d'un même domaine d'intérêt par l'utilisation d'axiomes différents ou équivalents pour définir des concepts ou utiliser des concepts totalement différents[2], par exemple la souris d'un ordinateur ou la souris l'animal (le rongeur).

1.6.4 hétérogénéité au niveau Semiotique

Concerne la manière dont les termes sont utilisés selon un contexte précis donc il peut avoir plusieurs interprétation selon le contexte ou domaine étudier. Ce type d'hétérogénéité est difficile a détecter par l'ordinateur et plus difficile a résoudre car c'est hors de son domaine.[2] par-exemple, dans le contexte maladie, une AAA signifie un abcès apical aiguë et dans le contexte du jeux-video un AAA définit une jeux produit par une grandes entrepris.

1.6.5 Terminologie

Dans cette section nous allons donner un glossaire avec les définitions et les termes qui vont être utiles pour la suite [2].

- **Matching** : Est le processus qui permet de trouver des correspondances entre les entités sur des ontologies différentes.
- **Correspondance** : Est la relation entre deux entités de différentes ontologies, ces entités peuvent être des concepts, des instances ou des propriétés.
- **Alignement** : Est un ensemble de correspondances entre deux ou plusieurs ontologies. L'alignement est le résultat du processus de matching.
- **Mapping** : C'est une version orientée de l'alignement, elle mappe une entité d'une ontologie à une autre entité d'une autre ontologie. Elle est conforme à la définition mathématique et non définie comme une relation générale. Une définition mathématique du mapping dit que l'objet mappé est égal à son image.
- **Matchers** : Les matchers sont des fonctions de similarité/dissimilarité entre deux entités. Le processus du matching permet la combinaison de ces fonctions.

1.7 Notion d'alignement

L'alignement est le processus de découverte des correspondances entre deux ou plusieurs ontologies différentes, entre les bases de données différentes ou entre les fichiers XML différents.

1.7.1 Processus d'alignement

Le processus d'alignement prend en entrée deux ou plusieurs ontologies et donne en sortie un alignement A entre les ontologies, d'autre paramètres peuvent être pris en compte (Figure 1.4) [2].

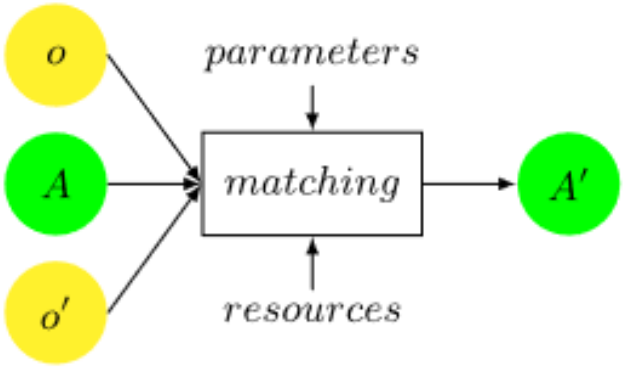


Figure 1.4: Processus d'alignement d'ontologies [2]

- Deux ontologies O et O'.
- Un alignement initial A a affiner avec le processus.
- Des ressources externes r tel que les dictionnaires.
- Des paramètres p tel que des seuils de poids.

1.7.2 Définition de Correspondance

Le processus d'alignement des ontologies donne comme résultat un alignement qui représente un ensemble de correspondance. Dans cette section nous allons donner la définition d'une correspondance.

Definition 1.7.1. Soient O et O' deux ontologies avec un langage d'entité associé Q_L et $Q_{L'}$, et un ensemble de relations d'alignement θ, une correspondance est un quaduplet [2] : La correspondance $< e, e', r, n >$ dit qu'il existe une relation r entre e et e'. tel que,

- e et e' deux entités des ontologies O et O'.
- r est une relations sémantique entre e et e' qui peut être une équivalence ($\equiv$), etc.
- n est une mesure de confiance, typiquement une valeur comprise dans [0,1].

Exemple d'une correspondances

une equivalence entre la classe [2] "$http://book.ontologymatching.org/example/culture-shop.owlBook$" et la classe "$http://book.ontologymatching.org/example/library.owlVolume$"

$$Book \equiv_{.85} Volume \tag{1.1}$$

1.8 Exemple d'alignement entre deux ontologies

La Figure (1.5) ci-dessous montre un exemple d'alignement de deux ontologies [2], ou leurs correspondances sont découvertes par un système d'alignement et sont représentées par des flèches en gras illustrant des relations d'équivalences, par exemple la correspondance entre "id" de "product" et "isbn" de "volume" possède un degré de confiance de "0.9", les autres flèches sont des relations de subsomption avec aussi un degrés de confiance.

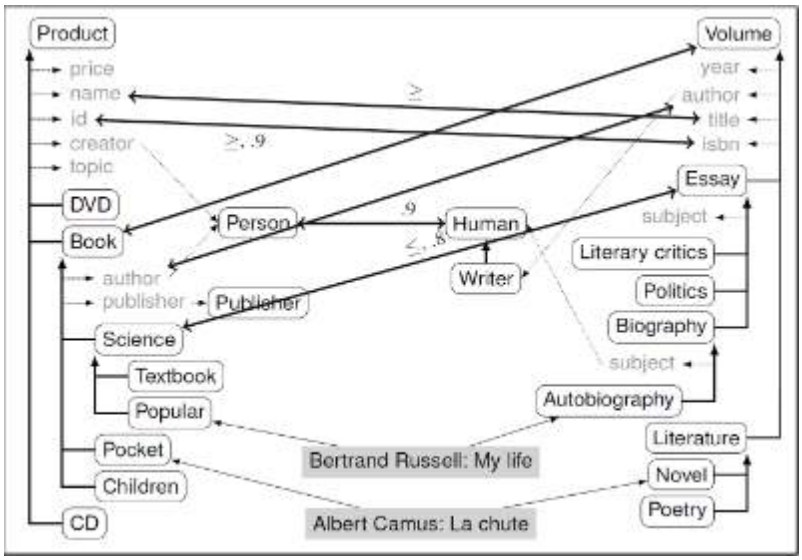


Figure 1.5: Exemple d'Alignement de Deux Ontologies [2]

1.9 Les Mesures de Similarité / Dissimilarité

Il y a plusieurs algorithmes pour calculer la similarité ou la dissimilarité entre deux entités de deux ontologies. Dans cette section nous allons présenter la définition de la similarité et la dissimilarité :

Definition 1.9.1. Similarité [2]. La similarité $\sigma : O \times O \rightarrow R$ est une fonction qui associe à une paire d'entités x et y un nombre réel exprimant la similitude entre x et y telle que :

$$\forall x, y \in O,\ \sigma(x, y) \geq 0\ (positivity) \tag{1.2}$$

$$\forall x \in O,\ \forall_{y,z \in o}, \sigma(x, x) \geq \sigma(y, z)\ (maximality) \tag{1.3}$$

$$\forall x, y \in O,\ \sigma(x, y) = \sigma(y, x)\ (symmetry) \tag{1.4}$$

Definition 1.9.2. Dissimilarité [2]. Soit un ensemble d'entité O, une dissimilarité $\delta : O \times O \rightarrow R$ est une fonction qui associe à une paire d'entités x et y un nombre réel entre x et y telle que :

$$\forall x, y \in O,\ \sigma(x, y) \geq 0\ (positiveness) \tag{1.5}$$

$$\forall x \in O,\ \sigma(x, x) = 0\ (minimality) \tag{1.6}$$

$$\forall x, y \in O,\ \sigma(x, y) = \sigma(y, x)\ (symmetry) \tag{1.7}$$

Definition 1.9.3. Distance [2]. Une distance (ou métrique) $\delta : O \times O \rightarrow R$ est une mesure de dissimilarité, la fonction de distance est vrai pour les deux propriétés suivantes :

$$\forall x, y \in O,\ \delta(x, y) = 0\ If\ and\ only\ if\ x = y\ (definiteness) \tag{1.8}$$

$$\forall x, y, z \in O,\ \delta(x, y) + \delta(y, z) \geq (x, z)\ (triangular\ inequality) \tag{1.9}$$

Il existe d'autres mesures de similarité pour plus de détails voir [2].

1.10 Techniques d'Alignement d'Ontologies

Dans l'alignement des ontologies les mesures citées ci-dessus peuvent être appliquées sur les chaînes de caractères (comme des noms, des labels, des descriptions), des valeurs (comme des cardinalités) ou sur des ensembles de chaînes de caractères (comme des ensembles d'instances). Dans cette section, nous allons voir trois type de mesures [2] :

- Les mesures basées sur les chaînes de caractères.
- Les mesures basées sur des tokens.
- Les mesures basées sur les ensembles.
- Les mesures basées sur les probabilités.

1.10.1 Techniques basées sur les chaînes de caractères

Les méthodes basées sur les chaînes de caractères permettent de calculer la similarité entre deux chaînes de caractères, ces méthodes prennent avantage de la structure d'une chaîne de caractère [2], par-exemple les classes "Book" et "Textbook" vont être similaires mais les classes Book et "Volume" ne le sont pas. Elles sont souvent utilisées pour faire correspondre les noms et la description des entités dans les ontologies. Il y a plusieurs techniques pour calculer la similarité / dissimilarité entre deux chaînes de caractères mais avant d'appliquer la comparaison de ces dites chaînes on doit faire un pré-traitement sur ces chaînes de caractères.

- **Transformation des majuscules en minuscules.** Convertit chaque lettre alphabétique en minuscules, par exemple, "Book" devient "book".
- **Suppression des accents.** Remplace chaque lettre alphabétique avec accent avec la lettre sans accent, par exemple, "Montréal" devient "Montreal".
- **Suppression de blanc.** Replace tous les caractères blanc comme des espaces, tabulassions ou une séquence des deux en un seul caractère blanc.
- **Suppression des liens.** Enlève certains liens entre les mots, par exemple, "peer-reviewed" devient "peer reviewed".
- **Suppression des valeurs numériques.** Supprime les nombres, par exemple, "book23545-8" devient "book".
- **L'élimination de la ponctuation.** Supprime les signes de ponctuations, par exemple, "C.D." devient "CD".

Égalité de chaîne de caractères

Definition 1.10.1. La méthode la plus simple pour comparer deux chaînes de caractère, retourne 0 si deux chaînes ne sont pas identiques, et retourne 1 si elles sont identiques, par définition [4] :

$$\sigma : S \times S \to [0\,1] \; tels \; que \; \forall x, y \in S, \sigma(x, x) = 1 \; et \; si \; x \neq y, \sigma(x, y) = 0. \tag{1.10}$$

Par exemple : "book" et "book" sont identique mais "book" et "textbook" ne l'ai pas et va retourner 0. Cette méthode n'est pas très utile vu que "book" et "textbook" sont similaires.

La distance de Hamming

Definition 1.10.2. [4] La distance de Hamming permet de compter les positions où les deux chaînes de caractères diffèrent, c'est une dissimilarité soit $S \times S \to [0\,1]$ tel que [2]:

$$\delta(s, t) = \frac{(\sum_{i=0}^{\min(|s|,|t|)} s[i] \neq t[i]) + ||s| + |t||}{\max(|s|, |t|)} \tag{1.11}$$

Notons que la distance de Hamming doit prendre en entrée deux chaînes de caractères de même longueur, ce qui n'est pas utile dans l'alignement des ontologies, car les classes et les instances n'ont pas la même longueurs, par exemple, "book" et "textbook" ne vont pas être comparés car "textbook" est plus long que "book", un autre exemple, "book" et "book" vont être similaire mais "book" et "look" ne le sont pas.

Similarité de sous chaîne de caractères

Definition 1.10.3. Soit $\delta : S \times S \to [0\,1]$ tel que $\forall x, y \in S$, et soit t la plus longue sous chaîne de caractère entre x et y alors [2] :

$$\delta(x, y) = \frac{2|t|}{|x| + |y|} \tag{1.12}$$

Par exemple, la similarité entre "article" et "aricle" est de $8/13 = 0.61$, alors que ces deux chaines ne sont pas similaires. La similarité entre "article" et "paper" est de $1/12 = 0.08$, alors qu'elles sont similaires.

Similarité n-Gram

Soit n-gram(s,n), l'ensemble de sous chaines de s de taille n. La similarité n-gram est une similarité $\sigma : S \times S \to R$ tel que [2] :

$$\sigma(x, y) = \frac{|ngram(s, n) \cap ngram(t, n)|}{min(|s|, |t|) - n + 1} \tag{1.13}$$

Cette fonction est très utile quand quelque caractères manquent. Par exemple, la similarité n-Gram entre "article" et "aricle" est de 0.5 alors que la similarité entre "article" et "paper" est de 0, et entre "article" et "particle" est de $5/6 = 0.83$.

Similarité Edit Distance

Definition 1.10.4. [4] Soit un ensemble d'opérations sur une chaîne de caractère Op tel que $op : S \rightarrow S$, et un coût $w : Op \rightarrow R$ tel que pour chaque paire de chaînes de caractère, il existe une séquence d'opérations qui permet de passer de la première chaîne à la deuxième et vice versa. L'edit distance est une dissimilarité $\delta : S \times S \rightarrow [0, 1]$ ou $\delta(x, y)$ est le coût de la séquence d'opérations la moins coûteuse qui transforme s en t :

$$\delta(x, y) = \min_{(op_i)_1; op_n(...op_1(s))=t} \sum_{i \in I} wop_i \tag{1.14}$$

Les opérations sont l'insertion d'un caractère ins(c,i), la substitution d'un caractère avec un autre $sub(c_1, c_2, i)$ et enfin l'opération effacer un caractère del(c,i). La distance de Leivenstein est le minimum d'opérations entre l'insertion, la substitution et l'effacement d'un caractère pour transformer une chaîne de caractère en une autre. Figure (1.6) montre la similarité entre les classes avec la similarité d'edit distance.

	Science	Children	Book	Person	DVD	Textbook	Product	Pocket	Publisher	Popular	CD
Politics	0.75	1.00	0.88	0.88	1.00	1.00	0.75	0.75	0.67	0.75	1.00
Thing	0.71	0.75	1.00	1.00	1.00	0.88	1.00	1.00	0.89	1.00	1.00
Autobiography	0.92	0.85	0.85	0.92	1.00	0.85	0.92	0.92	0.85	0.85	1.00
Novel	0.86	0.88	0.80	1.00	1.00	1.00	0.86	0.67	0.89	0.71	1.00
Biography	1.00	0.89	0.78	0.89	1.00	1.00	0.89	0.89	1.00	0.89	1.00
Writer	0.86	0.75	1.00	1.00	1.00	0.88	0.86	0.83	0.67	0.86	1.00
Essay	1.00	1.00	1.00	0.83	1.00	1.00	1.00	1.00	0.89	0.86	1.00
Volume	0.86	0.75	0.83	1.00	1.00	1.00	0.71	0.83	0.78	0.71	1.00
LiteraryCritic	0.93	0.93	1.00	0.86	1.00	0.93	0.86	0.93	0.93	0.86	0.93
Poetry	0.86	0.88	0.83	0.83	1.00	0.88	0.71	0.67	0.89	0.71	1.00
Literature	0.80	0.90	1.00	0.80	1.00	0.90	0.80	0.90	0.90	0.80	1.00
Human	0.86	0.88	1.00	0.83	1.00	1.00	1.00	1.00	0.89	0.71	1.00

Figure 1.6: La distance de Levenstein arrondie entre les labels des classes des ontologies[2]

Par exemple, Pocket et Novel d'après la distance de Levenstein sont très proches mais ces noms sont relativement distant entre eux.

Mesure de jaro

Definition 1.10.5. [4] La mesure de jaro est une dissimilarité $\sigma : S \times S \rightarrow [0, 1]$ tel que :

$$\sigma(s, t) = \frac{1}{3} \times (\frac{m}{|s|} + \frac{m}{|t|} + \frac{m - |transp(s, t)|}{m}) \tag{1.15}$$

avec $s[i] \in m$ si et seulement si $\exists_j \in [i - min(|s|, |t|)/2i + min(|s|, |t|)/2$ et transp(s, t) sont les éléments de m qui apparaissent dans un ordre différent dans s et t.

1.10.2 Techniques basées sur les tokens

Definition 1.10.6. [2] Les techniques basées sur les jetons considèrent les chaînes de caractères comme plusieurs mots (qui sont appelés sacs à mots), où un mot en particulier peut apparaître plusieurs fois. Ces approches fonctionnent bien quand les textes sont longs. ces techniques peuvent être utilisées sur les entités des ontologies en :

- En agrégeant différentes sources de chaînes de caractères : identifiant, labels, commentaires, documentation.
- Ou en divisant les chaînes de caractères en plusieurs tokens par-exemple **thèse-doctorat** devient **thèse et doctorat**.

Similarité cosinus

Definition 1.10.7. Soit $\vec{s}$ et $\vec{t}$ des vecteurs correspondant aux chaînes de caractères s et t dans un espace de vecteur V, la similarité cosinus est une fonction σV : $V \times V \rightarrow [0\ 1]$ tel que [2] :

$$\sigma V(s,t) = \frac{(\sum_{i \in |V|} \vec{s_i} \times \vec{t_i})}{\sqrt{\sum_{i \in |V|} \vec{s_i^2}} \times \sum_{i \in |V|} \vec{t_i^2}} \tag{1.16}$$

Similarité TFIDF

Definition 1.10.8. [2] La mesure TFIDF (Term frequency-Inverse document frequency) est utilisée pour voir la pertinence d'un document, par exemple, un sac à mots a un terme en tenant compte de la fréquence d'apparition du terme dans le corpus. Soit un corpus C, on définit la mesure suivante :

$$\forall t \in S,\ \forall s \in C\ tf(t,s) = t \sharp s\ (terme\ frequency) \tag{1.17}$$

$$\forall t \in S,\ idf(t) = log(\frac{|C|}{|s \in C;\ t \in s|}\ (inverse\ document\ frequency) \tag{1.18}$$

$$TFIDF(s,t) = tf(t,s) \times idf(t)\ (TFIDF) \tag{1.19}$$

Beaucoup de systèmes utilisent des mesures basées sur le TFIDF. Ces mesures calculent pour chaque termes sa pertinence par rapport au corpus, puis ils utilisent des techniques d'espaces vectoriels pour calculer la distance entre deux chaînes de caractères.

Similarité Kullback—Leiberdivergence

Definition 1.10.9. [2] Soit un ensemble de documents D avec la probabilité de distribution π et un ensemble de topics T, la divergence de Kullback-Leiber entre deux documents e et e' est :

$$\delta(e,e') = \sum_{t \in T} \pi(t|e) \times log_2(\frac{\pi(t|e)}{\pi(t|e')}) \tag{1.20}$$

1.10.3 Les mesures basées sur les ensembles

Le moyen le plus facile de comparer des concepts est de voir s'ils partagent les mêmes instances en testant l'intersection de leurs instances est de considérer ces concepts similaires. Si $A \cap B = A = B$, le problème est que un petit nombre de données erronées peut mener à de fausses conclusions.

La mesure de Jaccard

Definition 1.10.10. [2] Soit deux ensembles A et B et soit P(X) la probabilité qu'une instance soit dans l'ensemble X, la similarité de Jaccard est définie comme :

$$\sigma(A, B) = \frac{P(A \cap B)}{P(A \cup B)} \tag{1.21}$$

La distance de Hamming

Definition 1.10.11. [2]La distance de Hamming entre deux ensembles est une fonction de dissimilarité $\delta : 2^E \times 2^E \to R$ tel que $\forall_{x,y} \subseteq E$:

$$\delta(x, y) = \frac{|x \cup y| - |x \cap y|}{|x \cup y|} \tag{1.22}$$

1.10.4 Les mesures basées sur les probabilités

Les mesures basées sur les probabilités peuvent être utilisées pour améliorer un alignement initial. Les mesures de similarité ne tiennent pas en compte les inclusions logique entre les différents concepts des deux ontologies. Dans ce qui va suivre nous allons voir quelques méthodes de probabilités. dans la figure 1.7 un framework général de l'alignement avec un modèle probabiliste

Réseaux Bayésiens

Definition 1.10.12. [2] Les réseaux bayésiens est une approche probabiliste pour les modèles de cause à effet. Les réseaux bayésiens sont fait à partir d'un graphe acyclique dont les noeuds représentent les variables et les arcs entre les noeuds représentent les dépendances conditionnelles qui indiquent le sens de l'influence des noeuds. Par exemple un arc du noeud A appelé parent vers le noeud B appelé fils signifie que A a une influence directe sur B, un noeud qui influence un autre noeud est définit par une table de probabilité conditionnelle. En effet P(X|parents(X)) est la probabilité conditionnelle de la variable X, ou parents(X) est l'ensemble de tous les noeuds qui influencent directement X. La table de probabilité conditionnelle permet de construire la distribution de probabilité pour toutes les variables :

$$P(X_1, \ldots, X_n) = \prod_i P(X_i | parents(X_i)), i = 1, \ldots, n \tag{1.23}$$

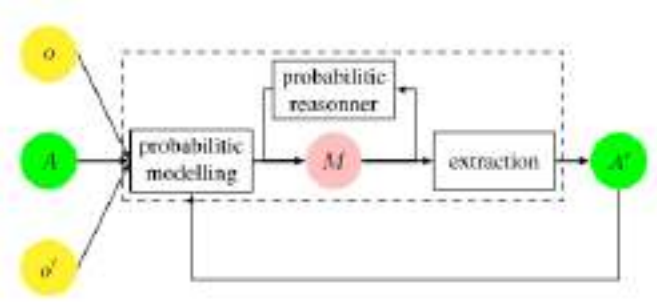


Figure 1.7: Paramètre général pour la correspondance d'ontologie probabiliste [2]

Les Réseaux de Markov et les Réseaux logiques de Markov

1. **Les Réseaux de Markov**

   Definition 1.10.13. [6] Les Réseaux de Markov sont des réseaux probabilistes structurés. Les Réseaux de Markov N = (V, E), sont faits à partir de noeuds V qui représentent des variables et des arrêtes E entre les noeuds qui représentent une dépendance statistique entre les variables. La distribution de probabilité dans un réseau de Markov est definie avec une fonction p sur des cliques C qui est un ensemble de noeuds Pc qui peut être associé avec chaque sous graphe complet dans le réseaux.

$$P(N) = \frac{1}{Z} \prod_{C \in cliques(N)} P_c(C) \tag{1.24}$$

   ou Z est une fonction de partition.

2. **Les Réseaux logiques de Markov**

   Definition 1.10.14. [6] Les Réseaux logiques de Markov unifie les Réseaux de Markov et la logique de première ordre. Dans ce réseau les noeuds représentent des formules atomiques et les cliques représentent grounding of formulas. Considérons un cas particulier de réseaux de Markov, tel qu'un modèle log-linéaire. Dans ce cadre, les potentiels sont remplacés par un ensemble de caractéristiques (formules) avec des poids. La distribution de probabilité du modèle log-linéaire est comme suit :

$$P(x) = \frac{1}{Z} \exp(\sum_i w_i f_i(x)) \tag{1.25}$$

   Z est une fonction de partition, w_i est un poids réel de la feature i et $f_i(x)$ est la i^{me} feature.

1.10.5 La méthode ProbaMap (Pi et Pc)

Definition 1.10.15. [8][3] Les méthodes PI et PC sont basées sur des taxonomies où chaque concept de la taxonomie peut être associé à un ensemble d'instances. Les mapping sont sous la forme d'inclusions de classe $\sqsubseteq$. Soit un mapping m sous la forme $A_i \sqsubseteq B_j$:

- La probabilité conditionnelle, dénotée $P_c(m)$, est définie comme $P_c(m) = Pr(B_j|A_i)$.
- La probabilité d'implication, dénotée $P_i(m)$, est définie comme $P_i(m) = Pr(\overline{A_i} \cup B_j$.

Théorème. Soit m : $C_i \sqsubseteq D_j$ un mapping entre deux taxonomies T_i et T_j. Soit O l'union entre les instances observées sur T_i et T_j. Soit $N = |O|$, $N_i = |\widehat{Ext}(C_i, O)|$, $N_j = |\widehat{Ext}(D_j, O)|$, $N_{ij} = |\widehat{Ext}(C_i \cup D_j, O)|$

$$\widehat{P}_i(m) = 1 + \frac{1 + N_{ij}}{4 + N} - \frac{1 + N_i}{2 + N} \tag{1.26}$$

$$\widehat{P}_c(m) = 1 + \frac{1 + N_{ij}}{4 + N} \times \frac{2 + N}{1 + N_i} \tag{1.27}$$

ou

- C_i : concepts de T_i.
- D_j : concepts de T_j.
- $\widehat{Ext}(C_i, O)$: Ensemble d'instances de O qui est reconnu par C_i.
- $\widehat{P}_i(m)$: L'estimation bayésienne de la probabilité d'implication
- $\widehat{P}_c(m)$: L'estimation bayésienne de la probabilité conditionnelle

1.10.6 La méthode PARIS

Definition 1.10.16. [7][3] La méthode PARIS permet de faire l'alignement automatique d'ontologie, elle permet de faire l'alignement d'instances mais aussi des relations et des classes. Dans ce projet nous allons considérer que l'alignement entre classes.
Une classe correspond à un ensemble d'entités, on peut être tenté de traiter les classes comme les instances et de trouver des équivalences entre elles, mais une structure de classe dans une ontologie peut être plus détaillée dans une autre ontologie. Ainsi, nous allons essayer de savoir si une classe d'une ontologie est une sous classe d'une classe d'une autre ontologie.

- **Concepts de fonctionnalité.** Une relation r est une fonction, si pour un premier argument donné, il y a un seul deuxième argument. Par exemple, une relation *estnéA* est une fonction, car une personne est né dans exactement un seul endroit. La relation est une fonction inverse, si son inverse est une fonction. La formule de la fonctionnalité d'une relation est comme suit :

$$fun(r, x) = \frac{\sharp x : \exists y : r(x, y)}{\sharp x, y : r(x, y)} \tag{1.28}$$

- **Le modèle probabiliste**

– **Équivalences des instances.** Nous voulons modéliser $P(x \equiv x')$, où l'instance x dans une ontologie est équivalente à une autre instance x' dans une autre ontologie. Assumons que les deux ontologies partagent la même relation r, nous voulons que la probabilité $P(x \equiv x')$ soit grande si r est hautement inverse fonctionnel, et s'il existe y, y' tels que r(x,y), r(x',y') (par-exemple x et x' partagent la même adresse mail), ceci peut être écrit en pseudo-formalisme comme suit :

$$\exists r, y, y : r(x, y) \wedge r(x, y) \wedge y \equiv y \wedge fun^{-1}(r) \; is \; high = x \equiv x \quad (1.29)$$

on transformant cette règle logique en affectation de probabilité pour $x \equiv x'$ comme suit :

$$P_1(x \equiv x') = 1 - \prod_{r(x,y), r(x',y')} (1 - fun^{-1}(r) * P(y \equiv y')) \quad (1.30)$$

En, d'autre termes, si on a une relation r avec $fun^{-1}(r) = 1$ avec r(x,y), r(x',y'), est $P(y \equiv y') = 1$ donc $P_1(x \equiv x') = 1$.

L'équation (1.31) ne tient compte que des preuves positives d'une égalité, et considère aussi des preuves contre une égalité, nous pouvons utiliser la modification suivante :
Nous voulons que la probabilité $P(x \equiv x)$ soit faible, s'il existe une relation hautement fonctionnelle r avec r(x, y) et si $y \neq y$ pour tout y avec r(x, y). Pseudo-formellement, cela peut être écrit comme suit:

$$\exists r, y : r(x, y) \wedge (\forall y' : r(x, y)) = y \neq y \wedge fun(r) \; is \; high = x \neq x \quad (1.31)$$

Qui peut être modélisé comme suit :

$$P_2(x \equiv x') = \prod_{r(x,y)} (1 - fun(r) \prod r(x', y')(1 - P(y \equiv y'))) \quad (1.32)$$

Dés qu'il existe une relation r avec fun(r) = 1 et avec r(x, y), r(x, y) et $P(y \equiv y) = 0$, donc $P2(x \equiv x) = 0$, nous combinons les deux probabilités en les multipliant, et ça nous donne ce qui suit :

$$P_3(x \equiv x') = P_1(x \equiv x') * P_2(x \equiv x') \quad (1.33)$$

– **Sous-relations.** S'il existe une relation r qui est commune entre les deux ontologies, le but est de trouver si $r \subseteq r'$ et la probabilité $P(r \subseteq r')$ est égale au nombre de paires dans r, qui sont des paires de r':

$$P(r \subseteq r') = \frac{\sharp x, y : r(x, y) \wedge r'(x, y)}{\sharp x, y : r(x, y)} \quad (1.34)$$

Le numérateur doit tenir compte des ressources qui ont déjà été appariés à travers les ontologies, par conséquent le numérateur est plus approprié et il est formulé comme suit :

$$\sharp x, y : r(x, y) \wedge (\exists x', y' : x \equiv x' \wedge y \equiv y' \wedge r(x', y')) \quad (1.35)$$

Il peut être modélisé comme suit :

$$\sum r(x,y)(1-\prod_{r(x',y')}(1-(P(x\equiv x')*P(y\equiv y')))) \tag{1.36}$$

Pour le dénominateur nous voulons normaliser le nombre de paires dans r qui ont une contrepartie dans l'autre ontologie, ceci peut s'écrire comme suit :

$$\sum r(x,y)(1-\prod_{x',y'}(1-(P(x\equiv x')*P(y\equiv y')))) \tag{1.37}$$

donc la probabilité final de $P(r\subseteq r')$ s'écrie sous la forme suivante :

$$\frac{\sum r(x,y)(1-\prod_{r(x',y')}(1-(P(x\equiv x')*P(y\equiv y'))))}{\sum r(x,y)(1-\prod_{x',y'}(1-(P(x\equiv x')*P(y\equiv y'))))} \tag{1.38}$$

Cette probabilité dépend de l'équivalence des deux instances.

– **Sous-classes.** Une classe peut-être vu comme un ensemble d'entités. On peut être tenté de traiter les classes comme des instances et chercher leurs équivalences. Mais la structure d'une classe d'une ontologie peut être plus détaillée qu'une classe d'une autre ontologie, c'est pour cette raison qu'au lieu de chercher l'équivalence entre les classes, Il faut voir si $c \subseteq c'$ et la probabilité $P(c \subseteq c')$ est la proportion des instances de c qui sont aussi des instances de c':

$$P(c\subseteq c') = \frac{\sharp(c\cap c')}{\sharp c} \tag{1.39}$$

De ce fait, on estime le nombre d'instances qui sont dans les deux classes :

$$E(\sharp c\cap c) = \sum x : type(x,c)(1-\prod_{y:type(y,d)}(1-(P(x\equiv y))) \tag{1.40}$$

Ensuite on divise le nombre attendu par le nombre total des instances de c :

$$P(c\subseteq c') = \frac{\sum x : type(x,c)(1-\prod_{y:type(y,d)}(1-(P(x\equiv y)))}{\sharp x : type(x,c)} \tag{1.41}$$

1.11 Conclusion

Dans ce chapitre, nous avons défini le web sémantique et les ontologies, ainsi que l'alignement et la notion de similarité et dissimilarité. Par la suite, nous avons cité quelques méthodes de similarité et dissimilarité.
Dans le chapitre suivant, nous allons présenter notre système **AlignFX**, dont l'objectif principal est la comparaison de plusieurs méthodes de similarités/probabilités, afin de prouver que les méthodes probabilistes offre de meilleurs résultats que les méthodes de similarités.

Conception de AlignFX

Chapitre 2

Conception de AlignFX

2.1 Introduction

2.1.1 Motivation

Les systèmes d'alignements extraient des correspondances entre les ontologies en utilisant les méthodes de similarités/probabilités. Il a été constaté que les méthodes de similarités sont beaucoup plus utilisées que les méthodes probabilistes malgré les résultats approximatifs de cette dernière, c'est pour cette raison que notre système a pour objectif la comparaison de plusieurs méthodes de similarités/probabilités afin de prouver que les méthodes probabilistes offrent de meilleurs résultats que les méthodes de similarités.

Dans ce qui va suivre, nous allons présenter à travers des exemples quelques problèmes auxquelles les méthodes de similarité sont dans l'incapacité de résoudre.

2.1.2 Problématique : Utilisation de langages naturels différents ou erreurs d'orthographe

Les méthodes de similarité (en particulier les mesures de distance) se concentrent surtout sur la terminologie des concepts, ainsi si ces concepts sont écrits différemment (Fautes d'orthographes, langages naturels différent etc...), ces méthodes donnent des résultats erronés

Figure 2.1: Exemple de problèmes liés aux erreurs d'orthographe

Dans la Figure 2.1 nous pouvons constater que l'ontologie 1 est remplie d'erreurs, ce qui va constituer un problème pour les méthodes de similarités. Par-exemple, **article et astite** sont équivalents mais à cause de cette erreur certaines méthodes de similarité ne peuvent pas trouver d'équivalence et donc elles ne peuvent pas extraire la correspondance entre ces deux concepts.

Figure 2.2: Exemple de problèmes s à différent langages naturels

Dans la Figure 2.2, nous pouvons constater que l'ontologie 1 est en français et que l'ontologie 2 est en anglais, cela va constituer un problème pour les méthodes de similarités. Par exemple, **Livre et Book** sont équivalents mais à cause de la différence de langue, certaines méthodes de similarités ne vont pas trouver d'équivalence et donc ne peuvent pas extraire la correspondance entre ces deux concepts. Vous pourrez dire que nous pouvons faire la traduction de l'ontologie du français en anglais ou l'inverse, mais le problème est que les traductions entre les langues sont parfois approximatives ou inexactes pour certaines.

Dans ce qui va suivre nous allons essayer de prouver que les méthodes probabilistes sont beaucoup plus efficaces et offrent de meilleurs résultats (trouver un alignement final complet/ correct) que les méthodes de similarités.

2.2 Architecture de AlignFX

Dans cette section nous allons présenter notre solution AlignFX en décrivant l'architecture du système proposé (Figure 2.3).

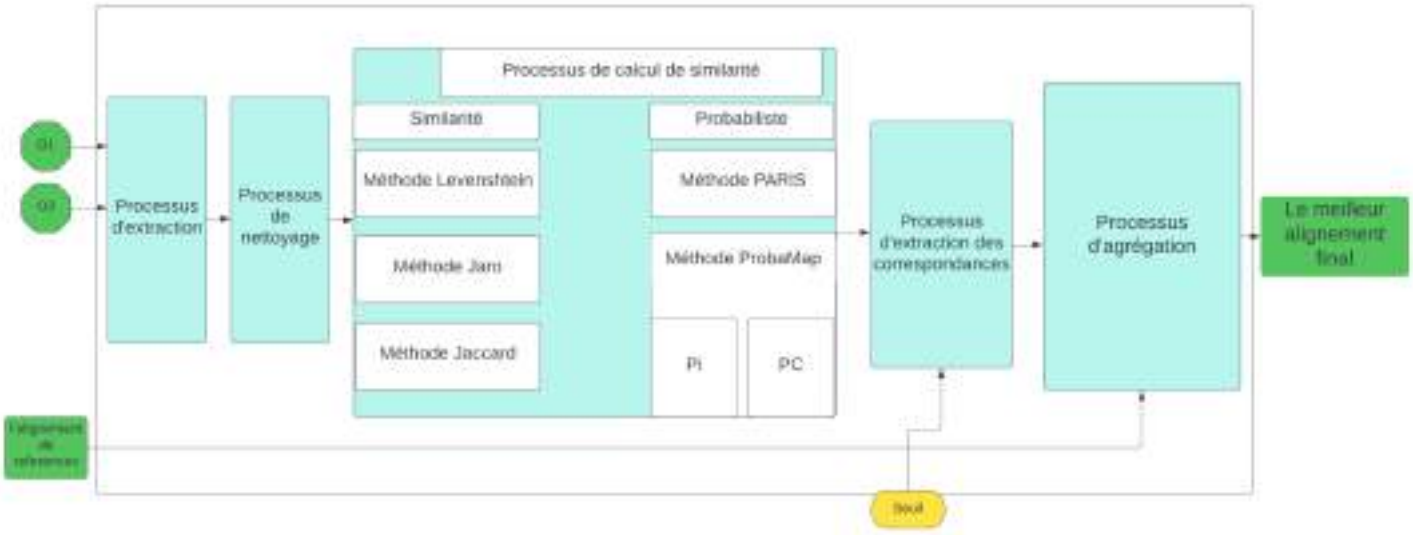


Figure 2.3: Architecture de AlignFX

Notre système considère en entré deux ontologies OWL O1 et O2, et un alignement de référence initial établi par un expert et donne en sortie un alignement final.

2.2.1 Processus de remplissage des ontologies

Les ontologies disponibles sur Internet ne contiennent pas beaucoup d'instances, y compris le site populaire OAEI, ces instances sont essentielles pour calculer la similarité/probabilité des correspondances d'où la nécessité de les remplir avec des données réelles. Pour cette raison, la première étape de notre travail consiste à remplir nos deux ontologies *O1* et *O2* à partir de *Dbpedia et de yago*. Dans ce qui suit nous allons détailler la méthode utilisée pour populariser les concepts de ces deux ontologies par des instances.

Yago

Yago est une base de connaissances avec des données réelles. Elle contient des entités et des relations entre ces entités. Yago contient plus de 50 million d'entités et 2 billion de faits. Il arrange ces entités en classes, par exemple, Paris appartient à la classes Pays. Ces classes sont arrangées en taxonomie, par exemple, la classe ville est une sous-classe de la classe lieu peuplé, cette classe est une sous-classe de la classe location géographique. Yago définit des relations entre les entités, par exemple, HabiteA est une relation entre personne et ville. Ces relations avec la taxonomie sont appelées une ontologie. Yago combine deux grandes ressources :

- **Wikidata** est une large base de connaissances dans le Web sémantique, elle possede un très bon dépôt d'entités, mais elle a une taxonomie difficile et les identifiants d'entités sont non lisibles par l'homme.
- **Schema.org** est une ontologie standard qui contient des classes et des relations, elle est maintenue par Google, mais cette ontologie ne contient aucune entité.

Ci-dessous un exemple de requête faite sur Yago :

```
PREFIX schema: <http://schema.org/>
PREFIX sc: <http://purl.org/science/owl/sciencecommons/>
PREFIX yago: <http://yago-knowledge.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x WHERE {
?x rdf:type schema:Article .
}
LIMIT 300
```

Dbpedia

DBpedia est une ontologie qui a été créé manuellement de Wikipedia en 2008. Elle contient 768 classes qui forme une hiérarchie de subsumption qui sont décrites par 3000 propriétés. L'ontologie de DBpedia contient 4.233.000 instances. Ci-dessus un exemple de requête faite sur DBpedia.

```
PREFIX schema: <http://schema.org/>
PREFIX sc: <http://purl.org/science/owl/sciencecommons/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x WHERE {
?x rdf:type schema:Article .
}
LIMIT 300
```

Une fois les requêtes faites, on aura les résultats des requêtes sparql sous forme RDF/XML. Ci-dessous un exemple de réponse à la requête SPARQL décrite ci-dessus :

$< rdf : Descriptions\ rdf : about = "http : //dbpedia.org/resource/GeorgeD.Tillman" >$

<gold:hypernym rdf:resource="http://dbpedia.org/resource/Article" />
</rdf:Descriptions> (2.1)

Dans rdf:about se trouve l'instance de type article qui est spécifié dans rdf:ressource.

L'algorithme 1 permet de remplir nos ontologies, il prend en entré l'ontologie OWL **O1** (respectivement O2) et un fichier **dos** écrit en RDF/XML qui contient les résultats des requêtes SPARQL, et donne en sortie l'ontologie **O1** (respectivement O2) remplie d'instances.

Algorithm 1 Remplissage des ontologies

Input : $O1, Dos$: O1 une ontologie OWL et Dos fichier résultat d'une requête sparql en rdf/xml.
Output : $O1$: O1 une ontologie OWL remplie d'instances.

1: $Qname \leftarrow dos.tags()$ ▷ contient tous les tags du fichier rdf
2: $attributes \leftarrow dos.tags().attributes$ ▷ contient les attributs des tags rdf, par exemple rdf:about
3: $m \leftarrow OWLManager.createOWLOntologyManager()$
4: $factory \leftarrow m.getOWLDataFactory()$ ▷ factory va servir à ajouter des instances
5: **if** $Qname.equalsIgnoreCase("rdf : Description")$ **then**
6: $about \leftarrow attributes.getValue("rdf : about")$
7: $arr \leftarrow about.replaceAll("http : //dbpedia.org/resource/","")$ ▷ Tableau qui contient toutes les instances d'un concept
8: $arrc \leftarrow attributes.getValue("rdf : ressource")$
9: **for** *each cls in* $O1.getClassesInSignature()$ **do**
10: **for** $i = 0; i < arrc.size(); i++$ **do**
11: **if** $cls.equalsIgnoreCase(ch)$ **then**
12: $result \leftarrow factory.getOWLNamedIndividual(IRI.create(O, +"#" + arr.get(i))$
13: $axiom \leftarrow factory.getOWLClassAssertionAxiom(cls, result)$
14: $addAxion \leftarrow new\ AddAxiom(O, axiom)$
15: $m.applyChange(addAxion)$
16: $m.saveOntology(O)$

- **Ligne 1-4** : Initialisation des variables.

- **Ligne 5-8** : Si les balises sont égales à <rdf:Description>, on récupère les instances de rdf:about et les concepts de rdf:ressources.

- **Ligne 9-16** : Pour chaque concept dans l'ontologie, et pour chaque concept récupéré de rdf:ressource, si les concepts sont égaux alors on remplit l'ontologie avec les instances récupérées précédemment puis nous sauvegardons l'ontologie.

2.2.2 Processus d'extraction

Une fois que les ontologies ont été remplies, l'étape suivante est l'extraction des concepts et des instances, afin de procéder au calcul de similarité/probabilité (Figure 2.4).
L'algorithme 2 montre l'extraction des concepts et des instances de l'ontologie O1 (respectivement O2).

Algorithm 2 Extraction des concepts et des instances

Input : $O1$: Une ontologie OWL.
Output: $arrM$: Tableau qui contient les concepts et les instances extraits.

1: $reasonerFactory \leftarrow new\ StructuralReasonerFactory()$ ▷ Pour créer un reasoner qui va permettre d'avoir les instances
2: $arrM \leftarrow \emptyset$ ▷ ArrayList d'un objet ClassInstanceMapping
3: **for** *each cls in* $O1.getClassesInSignature()$ **do**
4: $m \leftarrow newClassInstanceMapping()$ ▷ Instance de l'objet ClassInstanceMapping
5: $inst \leftarrow \emptyset$ ▷ Liste de chaînes de caractères
6: $m.setClassName(cls.getIRI().getShortForm())$
7: $reasoner \leftarrow reasonerFactory.createReasoner(O)$
8: $instances \leftarrow reasoner.getInstances(cls)$
9: **for** *each i in instances* **do**
10: $inst.add(i.getIRI().getFragment())$
11: $m.setInstances(inst)$
12: $arrM.add(m)$
return $arrM$

L'algorithme 2 prend en entré l'ontologie OWL **O1** (respectivement **O2**) et donne en sortie le tableau de ConceptInstanceMapping **arrM**.

- **Ligne 1-2**: Initialisation des variables.

- **Ligne 3-8** : Récupération de chaque concept dans l'ontologie O1 et la sauvegarde dans l'objet ClassInstancesMapping. Récupération aussi de ses instances.

- **Ligne 9-12** : Chaque instance de concept est sauvegardée dans l'objet ClassInstancesMapping.

Figure (2.4) illustre un exemple d'extraction de concepts et d'instances.

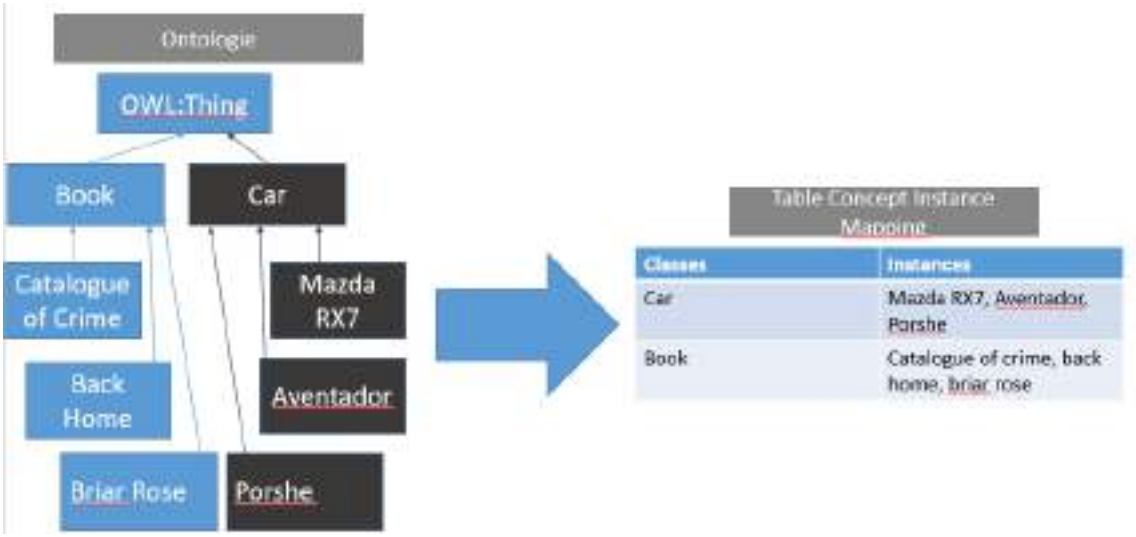


Figure 2.4: Exemple d'extraction de concepts et des instances

2.2.3 Processus de nettoyage

Le processus de nettoyage permet de nettoyer des concepts et des instances de l'ontologie, en effaçant les caractères spéciaux. Exemple :

$$Book100 - 50080 \implies book \tag{2.2}$$

L'algorithme 3 prend en entré une chaîne de caractère **c** et donne en sortie la chaîne de caractère nettoyée **normalisedtxt**.

Algorithm 3 Algorithme de nettoyage d'une chaîne de caractère

Input: *c*: La chaîne de caractère à nettoyer.
Output: *normalisedtxt*: La chaîne de caractère qui a été nettoyée

1: $normalisedtxt \leftarrow c.toLowerCase()$ ▷ Mettre la chaîne de caractère en minuscule
2: $normalisedtxt \leftarrow normalisedtxt.replace(" - ","")$ ▷ remplacer les "-" par des vides
3: $normalisedtxt \leftarrow normalisedtxt.replace("\_","")$ ▷ remplacer les "_" par des vides
4: $normalisedtxt \leftarrow normalisedtxt.replace(" ","")$ ▷ remplacer les espaces par des vides
5: $normalisedtxt \leftarrow removeAllDigit(normalisedtxt)$ ▷ effacer tous les nombres **return** *normalisedtxt*

- **Ligne 1-2**: Transformation de la chaîne de caractère en minuscule et suppression des tirets du six.

- **Ligne 3-5** : Suppression des tirets du huit, des espaces et des caractères numériques.

l'algorithme 3 est applicable juste sur une seule chaîne de caractère, alors que l'algorithme 4 va nettoyer les concepts et les instances (Section 2.3.2) obtenus du processus d'extraction.

Algorithm 4 Nettoyage des concepts et des instances d'une ontologie

Input: $arrC1$: Tableau qui continent les concepts et instances.
Output: $arrCNew$: Tableau qui contient les concepts et instances nettoyés.

```
1: arrCNew ← ∅                          ▷ Tableau qui va contenir les concepts et instances nettoyés
2: for each Cim in arrC1 do
3:     m ← newClassInstanceMapping()
4:     inst ← ∅                          ▷ Tableaux qui contiendra les instances nettoyées
5:     m.setClassName(Preprocess(Cim.getClassName()))
6:     for each ins in Cim.getInstances() do
7:         inst.add(Preprocess(ins))
8:         m.setInstances(inst)
9:     arrCNew ← arrCNew ∪ m
   return arrCNew
```

L'algorithme 4 prend en entré un tableau qui contient les concepts et les instances non nettoyés **arrC1** de l'ontologie **O1** (respectivement **O2**), et donne en sortie **arrCNew**, le tableau des concepts et des instances nettoyés. Un exemple d'application de l'algorithme est illustré dans la Figure (2.5).

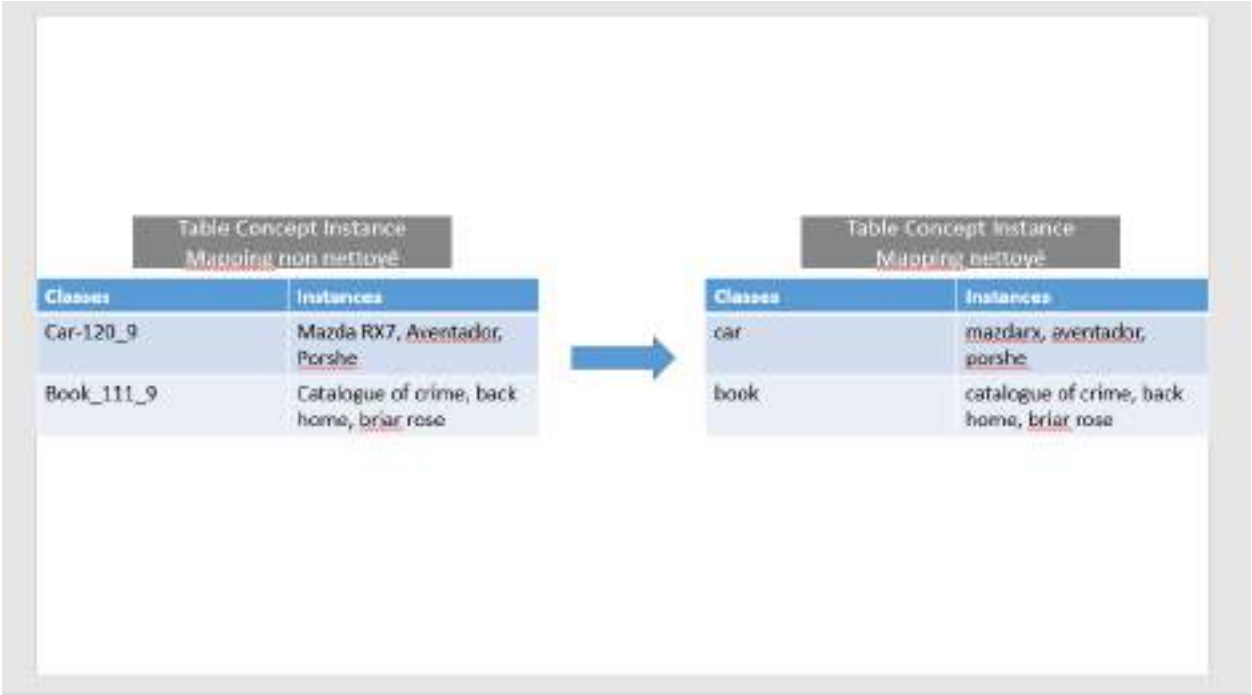


Figure 2.5: Exemple de nettoyage de concepts et d'instances

2.2.4 Processus de calcul des similarités

Le processus de calcul de similarité permet de calculer les valeur de confiances des correspondances en utilisant les méthodes de similarités/ probabilités entre deux concepts. Pour les méthodes de similarités nous avons choisi celle de Jaro, Levenshtein (section 1.10.1) et de Jaccard (section 1.10.3) . Pour les méthodes probabilistes nous avons opté pour la méthode PARIS (section 1.10.6) et de ProbaMap (section 1.10.5).

Méthodes de similarité

Les méthodes de similarités sont utilisées depuis longtemps dans le domaine de l'alignement sémantique. Ces méthodes permettent de résoudre des problèmes d'hétérogénéité terminologique, et retournent des correspondances avec des relations d'équivalences ≡.

- **Méthode de Jaro**

  L'algorithme 5(l'algorithme de Jaro) (section 1.10.1), est une méthode qui ce base sur la comparaison entre deux concepts.

Algorithm 5 Méthode de Jaro

Input: $c1$, $c2$: deux chaînes de caractères.
Output: sim: une similarité.

```
1:  if c1.lenght() && c2.lenght() == 0 then
2:      return 1;                          ▷ Si les chaînes sont vides, fin de l'algorithme
3:  matchDistance ← Integer.max(c1.lenght(), c2.lenght())/2   ▷ Pour avoir la distance maximal pour correspondre deux caractères
4:  matches ← 0                            ▷ le nombre de caractères correspondant
5:  transposition ← 0                      ▷ le nombre de transposition
6:  for i = 0; i < c1.lenght(); i++ do
7:      start ← Integer.max(0, i − matchDistance)
8:      end ← Integer.min(i + matchDistance + 1, i − c2.lenght())
9:      for j = start; j < end; j++ do
10:         if c1.charAt(i) != c2.charAt(j) then
11:             transpositions++
12:         matches++
13:         break
        return 1/3 ( matches/c1.lenght() + matches/c2.lenght() + (matches − transpositions)/matches )
```

L'algorithme 5 prend en entré deux chaînes de caractères **c1** et **c2** et donne en sortie une similarité.

- **Ligne 1-5**: initialisation des variables.

- **Ligne 6-12** : Parcourir la première et la deuxième chaîne de caractère, si deux caractères ne sont pas égaux alors incrémenter les transpositions sinon incrémenter le nombre de caractères correspondants.

Cet algorithme est applicable juste sur deux concepts seulement, mais pour le calcul de similarité entre les concepts obtenus à la sortie de l'étape de nettoyage (Section 2.3.3), nous avons utilisé Jaro pour chaque paire de concepts dans l'algorithme 6.

Algorithm 6 Algorithme de Jaro appliqué sur un tableau de concept

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: mps: un tableau de correspondance.

$mps \leftarrow \emptyset$ ▷ Un tableau qui contiendra toutes les correspondances calculées
for *each* Cim *in* $arrC1$ **do**
  for *each* $Cim2$ *in* $arrC2$ **do**
    $mp \leftarrow new\ Mapping()$ ▷ Cré une correspondance vide
    $mp.setSourceId(Cim.getClassName())$
    $mp.setTargetId(Cim2.getClassName())$
    $mp.setSimilarity(Jaro(Cim.getClassName(),\ Cim2.getClassName()))$
    $mps \leftarrow mps \cup mp$
return mps =0

L'algorithme 6 prend en entré deux tableaux, le ConceptInstanceMapping **ArrC1** de l'ontologie **O1** et **ArrC2** de l'ontologie **O2** et retourne un tableau de correspondances **mps**. Pour chaque pair de concepts dans ArrC1, ArrC2 on appelle l'algorithme de Jaro pour calculer la similarité de leur correspondance, à la fin on ressort avec un tableau mps contenant toutes les valeurs de similarité calculées.

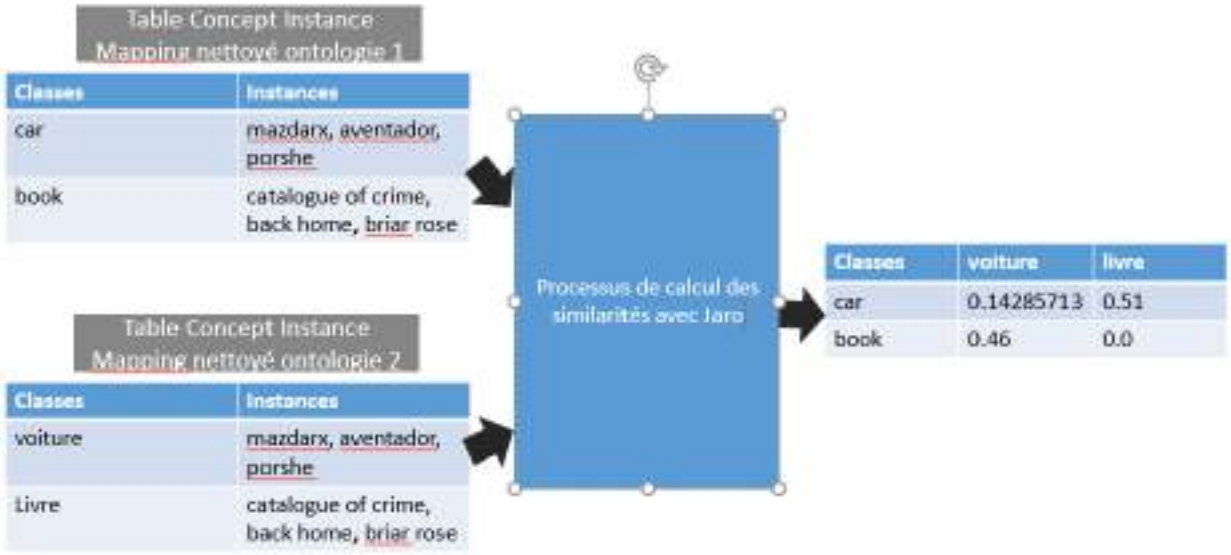


Figure 2.6: Exemple de calcul des similarités avec la méthode jaro

La figure (2.6) montre un exemple de calcul des correspondances avec la méthode Jaro entre quatre concepts, deux en français et deux en anglais. Nous pouvons voir que la similarité de jaro entre les concepts *car* et *voiture* est de zéro, malgré que les deux concepts sont équivalents, il est de même pour *book* et *livre*. Ceci est un exemple qui montre que dans certains cas les méthodes de similarités comme Jaro donne des résultats approximatifs.

- **Méthode de levenshtein**

L'algorithme 7 est celui de la méthode de Levenshtein (section 1.10.1) qui se base sur la comparaison de deux concepts comme la méthode de jaro.

Algorithm 7 Méthode de levenstein

Input: $c1$, $c2$: deux chaînes de caractères.
Output: arr: une distance en entier.

```
1:  c1Lenght ← c1.length()                          ▷ taille de la chaine de caractere c1
2:  c2Lenght ← c2.length()                          ▷ taille de la chaine de caractere c2
3:  arr[ ][ ] ← new int[c1Lenght + 1][c2Lenght + 1] ▷ initialiser un tableau a deux dimensions avec
    les taille c1Lenght et c2Lenght
4:  for (i = 0; i <= c1Lenght ; i++) do
5:      for (j = 0; j <= c2Lenght ; j++) do
6:          if i == 0 then
7:              arr[i][j] = j
8:          else if j == 0 then
9:              arr[i][j] = i
10:         else if c1.charAt(i − 1) == c2.charAt( j − 1) then
11:             arr[i][j] ← arr[i − 1][j − 1])
12:         else
13:             arr[i][j] ← 1 + Min(arr[i][j − 1], arr[i − 1][j], arr[i − 1][j − 1])
    return arr[c1Lenght][c2Lenght]
```

L'algorithme 7 prend en entrée deux chaînes de caractères **c1** et **c2** et retourne une distance **arr**.

- **Ligne 1-3**: Initialisation des variables.

- **Ligne 4-9** : Parcours de la première et la deuxième chaîne de caractère, si la première chaîne de caractère est vide, on rempli le tableau avec j représentant le nombre d'insertion pour passer d'un caractère vide à la deuxième chaîne de caractère, sinon si la deuxième chaîne de caractère est vide alors on rempli le tableau avec i, qui représente le nombre de suppression pour passer de la première chaîne de caractère à un caractère vide.

- **Ligne 10-13** : Sinon si les caractères entre la première et la deuxième chaîne sont équivalents alors on prend le résultat précèdent, Sinon on considère l'opération qui a le moins de coût entre l'insertion, la suppression ou le remplacement d'un caractère.

L'étape suivante consiste à normaliser la valeur de levenshtein :

Algorithm 8 Normaliser les valeurs de la méthode levenshtein

Input: $c1$, $c2$: deux chaînes de caractères.
Output: *result*: une similarité.

$result \leftarrow 0$
$temp \leftarrow editDistance(c1, c2)$
if $c1.lenght() >= c2.lenght()$ **then**
    $result \leftarrow 1 - \frac{temp}{c1.lenght()}$
else
    $result \leftarrow 1 - \frac{temp}{c2.lenght()}$
return *result*

L'algorithme 8 prend en entré deux chaînes de caractères **c1** et **c2**, et compare leurs tailles. Si la première est plus grande alors en calcule la similarité en utilisant la première chaîne de caractère sinon on calcule la similarité en utilisant la deuxième.

Mais encore une fois, cet algorithme est applicable juste sur deux chaînes de caractère et nous avons besoin de calculer cette similarité entre plusieurs chaînes de caractères relatives à plusieurs concepts. Ainsi, nous proposons L'algorithme 9 en considérant les concepts et les instances obtenus à la sorties du processus de nettoyage (Section 2.3.3). Nous avons utilisé Levenshtein pour chaque paire de concepts.

Algorithm 9 Edit Distance appliqué sur un tableau

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: *mps*: un tableau de correspondances.

$mps \leftarrow \emptyset$ ▷ Un tableaux de correspondance qui contiendra toutes les correspondances calculées
for *each* Cim *in* $arrC1$ **do**
    for *each* $Cim2$ *in* $arrC2$ **do**
        $mp \leftarrow new\ Mapping()$ ▷ Crée une correspondance vide
        $mp.setSourceId(Cim.getClassName())$
        $mp.setTargetId(Cim2.getClassName())$
        $mp.setSimilarity(EditDistanceSimilarity(Cim.getClassName(), Cim2.getClassName()))$
        $mps \leftarrow mps \cup mp$
return *mps*

L'algorithme 9 prend en entré deux tableaux, ConceptInstanceMapping **ArrC1** de l'ontologie **O1** et **ArrC2** de l'ontologie **O2** et retourne un tableau de correspondances **mps**. Pour chaque pair de concepts dans ArrC1, ArrC2 on appelle l'algorithme de Edit distance pour calculer la similarité de leurs correspondances. A la fin on ressort avec un tableau mps contenant toutes les valeurs de similarité calculées.

La figure (2.7) montre un exemple de calcul des correspondances entre quatre concepts deux en français et deux en anglais. Nous pouvons voir que la similarité de Levenshtein entre les concepts *car* et *voiture*, malgré qu'ils soient équivalents, a donné une similarité de 0.147, et pour *book* et *livre* nous avons obtenu une similarité de zéro. Ainsi, les méthodes jaro et levenshtein ont le même problème.

- **Méthode de jaccard**

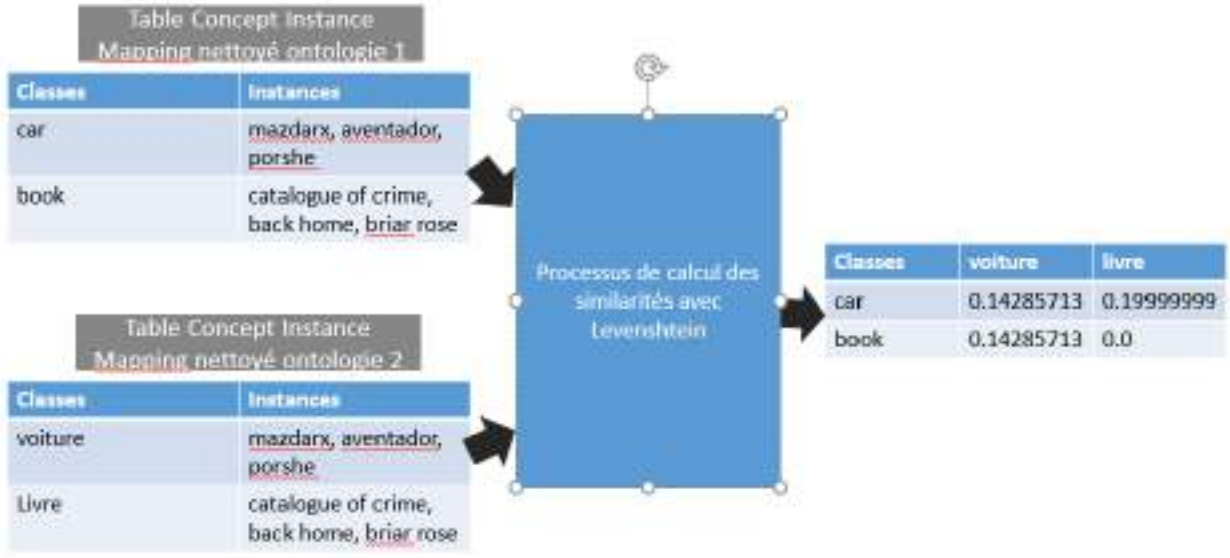


Figure 2.7: Exemple de calcul des similarités avec la méthode levenshtein

L'algorithme 10 est celui de la méthode de Jaccard, cette dernière se base sur la comparaison des concepts en utilisant les instances.

Algorithm 10 Méthode de Jaccard

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: mps: un tableau de correspondances.

$similarity \leftarrow 0$
$mps \leftarrow \emptyset$ ▷ Un tableaux de correspondance qui contiendra toutes les correspondances calculées
for *each* Cim *in* $arrC1$ **do**
  for *each* $Cim2$ *in* $arrC2$ **do**
    $mp \leftarrow new\ Mapping()$
    $mp.setSourceId(Cim.getClassName())$
    $mp.setTargetId(Cim2.getClassName())$
    $similarity \leftarrow \frac{cim.getInstances() \cap cim2.getInstances}{cim.getInstances() \cup cim2.getInstances()}$
    $mp.setSimilarity(similarity)$
    $mps \leftarrow mps \cup mp$
return mps

La figure (2.8) montre un exemple de calcul des correspondances avec la méthode Jaccard, en considérant toujours le même exemple. Comme la similarité de jaccard est basée sur les instances, cette méthode résout le problème des deux méthodes vues précédemment. En effet la similarité calculée entre les concepts *car* et *voiture* est de 0.98, et celle de *book* et *livre* est de 0.88. Ainsi, la méthode jaccard est plus efficace mais le problème est quelle extrait des correspondances avec des relations

d'équivalence, ce qui provoque une perte de correspondances importante avec les relations d'inclusion ou de généralisation.

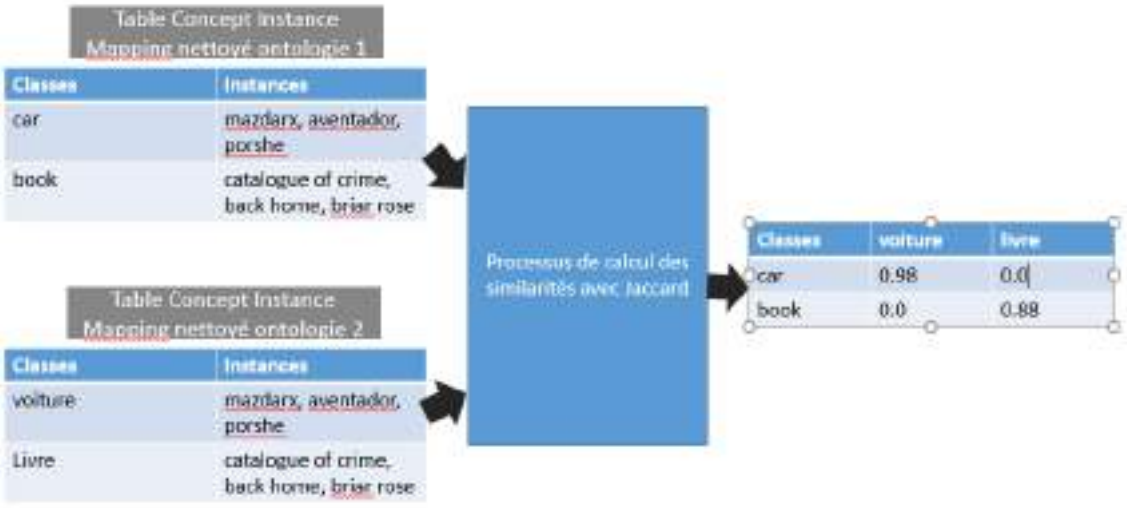


Figure 2.8: Exemple de calcul des similarités avec la méthode Jaccard

Méthodes Probabilistes

Les méthodes probabilistes sont utilisées pour résoudre plusieurs types d'hétérogénéités (Conceptuel, Sémiotique etc...), elles retournent des correspondances avec des relations d'inclusion $\sqsubseteq$. Dans ce qui va suivre, nous allons présenter deux méthodes , la méthode PARIS et la méthode ProbaMap.

- **Méthode PARIS** Cette méthode permet de faire un alignement d'instances, de concepts et des relations. Mais dans notre travail nous avons considérer que l'alignement des concepts avec la comparaison de leurs ensembles d'instances. Ceci est illustré par l'algorithme 11.

Algorithm 11 Méthode PARIS

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: mps: un tableau de correspondances.

$probability \leftarrow 0$
$mps \leftarrow \emptyset$ ▷ tableau de correspondances calculées
for *each* c *in* $arrC1$ **do**
  for *each* c' *in* $arrC2$ **do**
    $P(c \subseteq c') = \frac{\sharp(c \cap c')}{\sharp c}$
    $mp \leftarrow new\ Mapping()$ ▷ Créé une correspondance vide
    $mp.setSourceId(c)$
    $mp.setTargetId(c')$
    $mp.setSimilarity(P(c \subseteq c'))$
    $mps \leftarrow mps \cup mp$
return mps

L'algorithme 11 prend en entré deux tableaux de ConceptInstanceMapping **ArrC1** de l'ontologie **O1** et **ArrC2** de l'ontologie **O2** et retourne un tableau de correspondances **mps**.

- **Ligne 1-2**: Initialisation des variables.

- **Ligne 3-5** : Pour chaque ConceptInstanceMapping contenu dans le premier et le deuxième tableau nous récupérons leurs instances, et nous calculons la probabilité de PARIS.

- **Ligne 6-10** : Création d'une nouvelle correspondance en remplissant la partie gauche et droite de cette dernière et la similarité entre les deux concepts comparés. La nouvelle correspondance est ajoutée dans le tableau des correspondances.

Table des correspondances de Paris

SourceId	TargetId	Probabilité	Type de relations
art	entertainement	0.666	⊆
ordinateur	computer	0.909	⊆
logiciel	software	0.904	⊆
jeux	entertainement	0.952	⊆

Figure 2.9: Exemple de correspondance avec la méthode Paris

La figure (2.9) montre les correspondances obtenues avec la méthodes de PARIS. Comme il a été mentionné précédemment, la méthode Jaccard donne de bon résultats mais avec une perte importante de correspondances due au fait que jaccard ne génère que des correspondances avec des relations d'équivalence, alors que si nous utilisons les méthodes probabilistes il n'y a aucune perte de correspondances. Par exemple, avec la méthode PARIS nous avons trouvé des correspondances supplémentaires, comme entre *art* et *entertainement* avec une probabilité de 0.666.

- **Méthode ProbaMap**

  La deuxième méthode probabiliste est la méthode ProbaMap qui est composée de deux méthodes, la méthode PI et la méthode PC.

L'algorithme 12 montre le pseudo-algorithme de la méthode PI.

Algorithm 12 Méthode PI

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: mps: un tableau de correspondance.

$Pi \leftarrow 0$
$N \leftarrow getInstances(arrC1, arrC2)$
$TailleTotal \leftarrow N.size()$
for *each* C *in* $arrC1$ **do**
    for *each* D *in* $arrC2$ **do**
        $Ni \leftarrow |\widehat{Ext}(C_i, O)|$
        $Nij \leftarrow |\widehat{Ext}(C \cup D, N)|$
        $\widehat{P}_i(m) \leftarrow 1 + \frac{1+Nij}{TailleTotal+4} - \frac{1+Ni}{2+TailleTotal}$
        $mp \leftarrow new\ Mapping()$ ▷ Créé une correspondance vide
        $mp.setSourceId(C)$
        $mp.setTargetId(D)$
        $mp.setSimilarity(\widehat{P}_i(m))$
        $mps \leftarrow mps \cup mp$
return mps

L'algorithme 13 présente le pseudo-algorithme de la méthode PC.

Algorithm 13 Méthode PC

Input: $arrC1$, $arrC2$: arrC1 et arrC2 deux tableaux ConceptInstanceMapping.
Output: mps: un tableau de correspondance.

$Pc \leftarrow 0$
$N \leftarrow getInstances(arrC1, arrC2)$
$TailleTotal \leftarrow N.size()$
for *each* C *in* $arrC1$ **do**
    for *each* D *in* $arrC2$ **do**
        $Ni \leftarrow |\widehat{Ext}(C, O)|$
        $Nij \leftarrow |\widehat{Ext}(C \cup D, N)|$
        $\widehat{P}_i(m) \leftarrow 1 + \frac{1+Nij}{TailleTotal+4} - \frac{2+TailleTotal}{1+Ni}$
        $mp \leftarrow new\ Mapping()$ ▷ Créé une correspondance vide
        $mp.setSourceId(C)$
        $mp.setTargetId(D)$
        $mp.setSimilarity(\widehat{P}_i(m))$
        $mps \leftarrow mps \cup mp$
return mps

Les deux algorithmes présentés ci-dessus, donnent en sortie un ensemble de correspondances qui sont illustrées dans la figure (2.10).

De même que pour la méthode PARIS, les méthodes PI et PC peuvent trouver des correspondances importantes grâce à la relation d'inclusion des correspondances. fig(2.10)

L'algorithme ProbaMap globale est donné dans Algorithme 14.

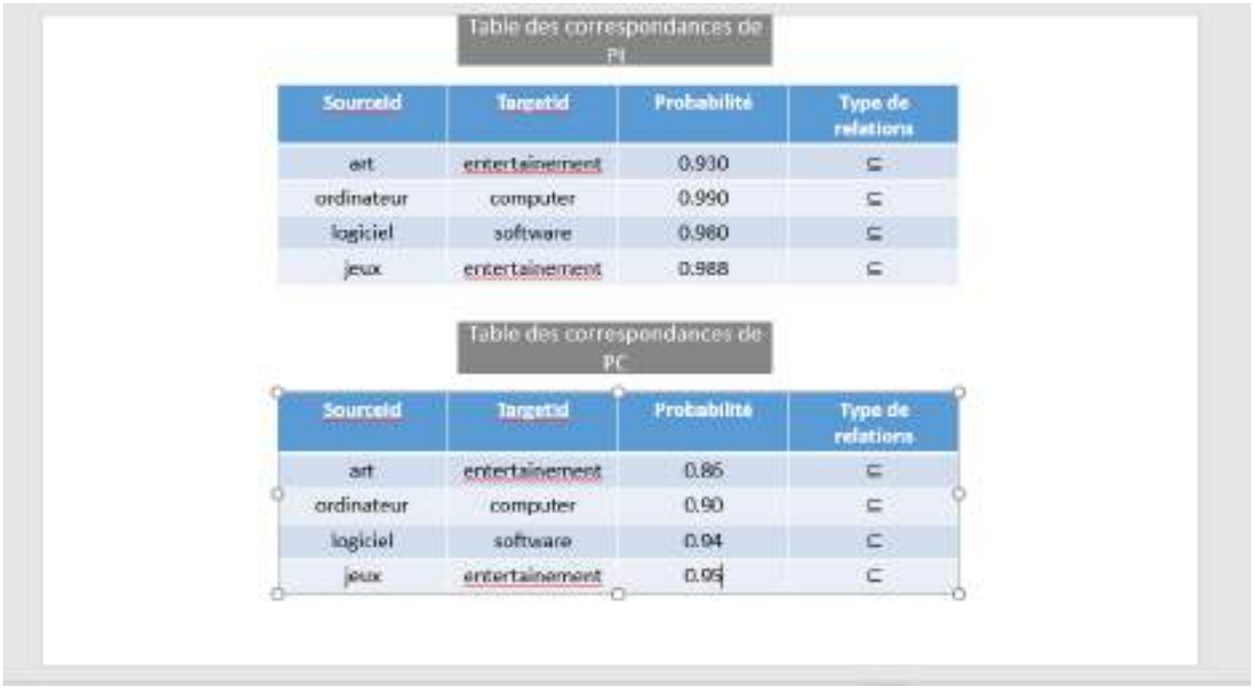


Figure 2.10: Exemple de correspondances avec les méthodes PI et PC

Algorithm 14 Méthode ProbaMap

Input: $arr1, arr2$: Tableaux de correspondances de PC et PI respectivement.
Output: crs: les correspondances de probaMap.

$crs \leftarrow \emptyset$
for $Cr \in arr1$ **do** ▷ correspondance dans arr1 de pc
  for $Cr2 \in arr2$ **do** ▷ correspondance dans arr2 de pi
    if $Cr.getSourceId() = Cr2.getSourceId()$ *And* $Cr.getTargetId() = Cr2.getTargetId()$ **then**
      $Crfinal \leftarrow \emptyset$ ▷ une correspondance vide
      $Crfinal.setSourceId(Cr.getSourceId())$ ▷ affecter la correspondance avec la partie gauche la même que dans Cr2
      $Crfinal.setTargetId(Cr.getTargetId())$ ▷ affecter la correspondance avec la partie droite la même que dans Cr2
      $sim \leftarrow \frac{Cr.getSimilarity()+Cr2.getSimilarity()}{2}$ ▷ Calculer la moyenne entre la probabilité de Cr1 et Cr2
      $Crfinal.setSim(sim)$ ▷ affecter la moyenne entre la probabilité de Cr1 et Cr2
      $crs \cup Crfinal$
return crs

L'algorithme 14 prend en entré deux tableaux de correspondances **Arr1** résultat de la méthode **PC** et **Arr2** résultat de la méthode **PI** et retourne un tableau de correspondances **crs**. Le tableau renvoyé aura la moyenne de Arr1 et Arr2, autrement dit la moyenne des résultats trouvés par PC et PI. Cet algorithme nous donne en sortie un ensemble de correspondances qui est illustré dans la figure (2.11).

Table des correspondances de ProbaMap

SourceId	TargetId	Probabilité	Type de relations
art	entertainement	0.79	⊆
ordinateur	computer	0.94	⊆
logiciel	software	0.93	⊆
jeux	entertainement	0.96	⊆

Figure 2.11: Exemple de correspondances avec la méthode ProbMap

2.2.5 Processus d'extraction des correspondances

Après l'accomplissement du processus de calcul de similarité/probabilité (avec les méthodes de similarités et des méthodes probabilistes), vient l'étape d'extraction des correspondances qui à pour but d'extraire l'alignement final, c'est à dire, sélectionner les correspondances qui feront partie de l'alignement et éliminer toutes les correspondances avec une similarité très faibles. Ce processus est présenté à travers l'algorithme 15.

Algorithm 15 Génération de l'alignement final

Input: $arr1$: une liste de correspondances calculées par une méthode.
$tresh$: un seuil.
Output: crs: Toutes les correspondances qui sont supérieures ou égales au seuil.

1: $crs \leftarrow \emptyset$
2: **for** $Cr \in arr1$ **do** ▷ Chaque correspondances dans arr1
3: **if** $Cr.getSimilarity() \geq tresh$ **then**
4: $crs \leftarrow crs \cup Cr$
return crs

L'algorithme 15 prend en entré un seuil **tresh** et un tableau de correspondances **arr1** généré, après application d'une des méthodes vues précédemment et retourne un tableau de correspondances **crs** qui comporte des correspondances qui sont au dessus du seuil défini.

- **Ligne 1**: Initialisation du tableau de correspondances.
- **Ligne 2-4**: Pour chaque correspondance du tableau, si la similarité est supérieure ou égale au seuil alors cette correspondance est retenue.

La figure (2.12) montre un exemple de l'exécution de cet algorithme.

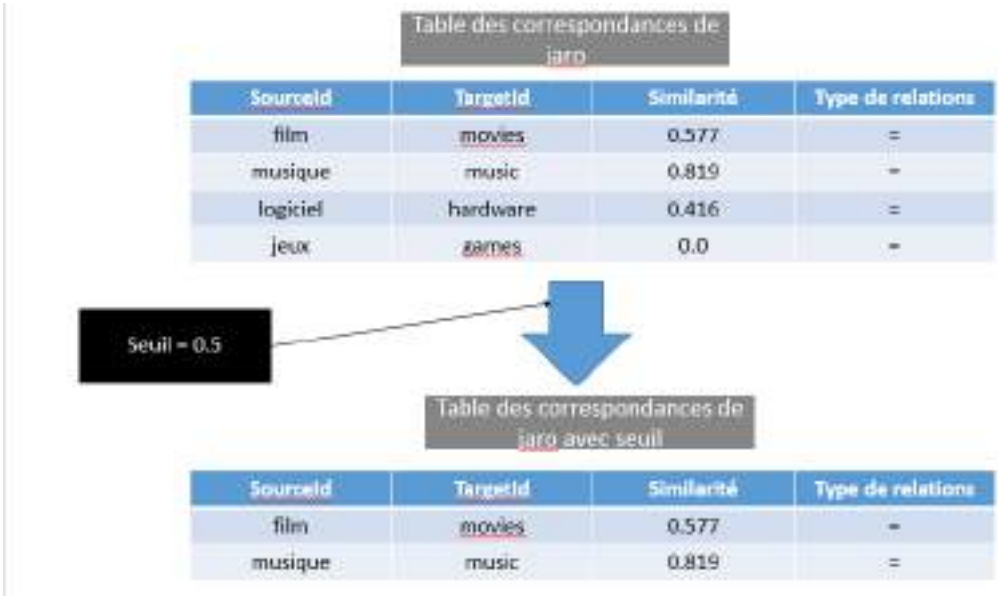


Figure 2.12: Résultat d'extraction des correspondances (Méthodes jaro)

2.2.6 Processus d'agrégation

Maintenant que les alignements finaux de toutes les méthodes sont générés, nous pouvons procéder à la dernière étape de notre systèmes qui est l'agrégation et la comparaison entre les méthodes. La Figure (2.13 montre un exemple de résultat après application de ce processus. Les etapes suivies sont comme suit :

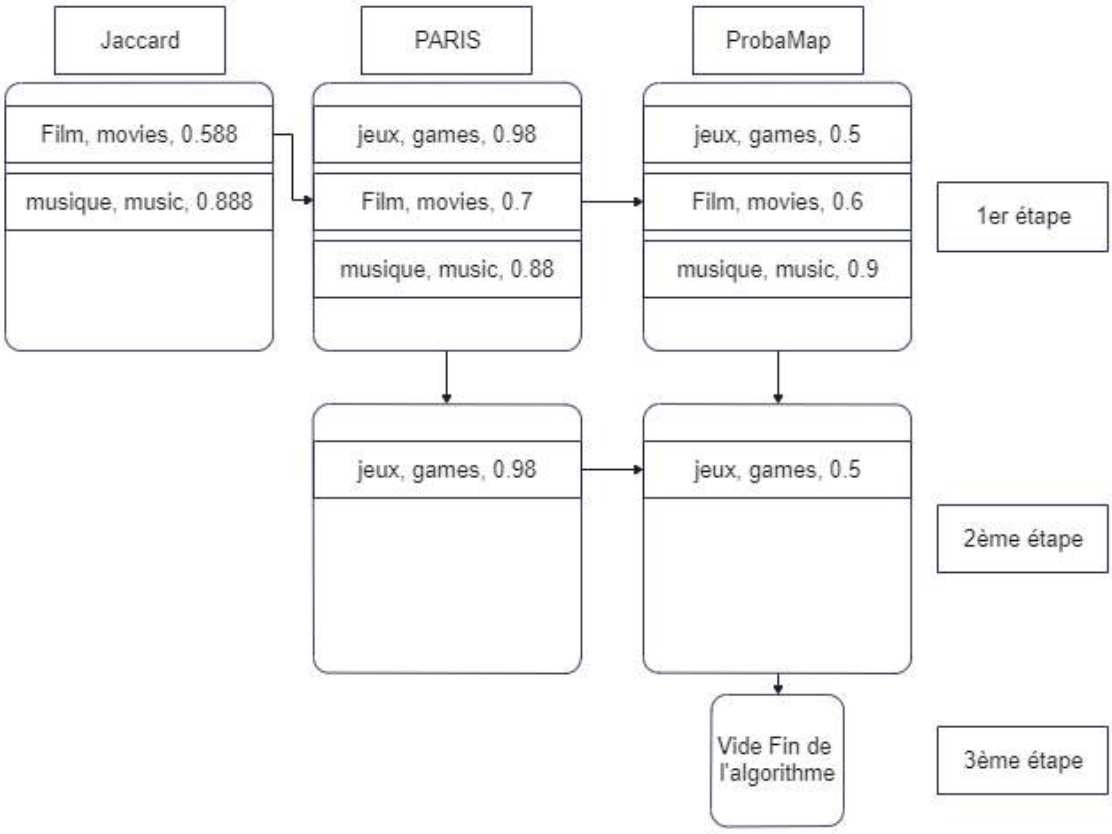


Figure 2.13: Extrait du résultat du processus d'agrégation/comparaison

1. Dans cette étape, nous allons trier les alignements finaux selon leurs tailles, par exemple, dans la figure (2.13) nous pouvons voir que la méthode Jaccard a le moins de correspondances, donc c'est elle qu'on va comparer en premier.

2. Ensuite, au lieu de comparer tous les alignements en même temps, nous allons comparer, par exemple, Jaccard avec Paris et voir s'ils ont les mêmes correspondances (même partie gauche et droite), alors on place la correspondance de paris dans une variable temporaire. On refait la même chose entre Jaccard et probaMap.

3. Une fois les variables temporaires remplies, nous allons appliquer différentes comparaisons, c'est à dire, le minimum, le maximun et moyenne entre par exemple, la correspondances de jaccard (Film, movies, 0.588), la variable temporaire de paris (Film, movies, 0.7) et la variable temporaire de Probamap (Film, movies, 0.6).

4. Une fois les comparaisons faites, chacune des comparaisons (min, max moyenne) aura son propre alignement, par exemple, l'alignement du min aura comme correspondance celle de jaccard, l'alignement du max celle de PARIS et l'alignement de la moyenne aura la correspondance (Film, movies,0.62) qui est la moyenne entre les troix correspondances de Jaccard (Film, movies, 0.588), PARIS (Film, movies, 0.7) et ProbaMap (Film, movies, 0.6).

5. Une fois le processus d'agrégation terminé, nous effaçons la correspondance (film,movies) de paris et probaMap, car nous n'avons plus besoin de les comparer par la suite.

6. Même chose pour (musique, music,0.888), on réitère l'étape N° 2 ou PARIS et ProbaMap ne contiennent pas les correspondances de l'alignement de Jaccard car nous n'avons plus besoin de ces correspondances puisqu'ils existent dans l'alignement final.

7. Une fois la dernière comparaison faite à l'étape 3 entre les deux alignement, l'algorithme s'achève.

Ci-dessous l'algorithme 16 qui décrit la comparaison et l'agrégation avec le Min.

Algorithm 16 La comparaison avec le Min

Input: *Align*: L'alignement final des méthodes
Output: *AlignMin*:Agrégation de tous les alignement en un seul selon la compraison du min

1: $AlignMin \leftarrow \emptyset$
2: $temp \leftarrow null$
3: $temp1 \leftarrow null$
4: $temp2 \leftarrow null$
5: $temp3 \leftarrow null$
6: $min \leftarrow 0$
7: $temp$, $temp1$, $temp2$, $temp3 \leftarrow Compare(Align[0], [Align[1], Align[2], Align[3], Align[4]])$
8: $min \leftarrow min(Align[0].getCorrespondance(), temp, temp1, temp2, temp3)$
9: $AlignMin \cup min$
10: $temp1$, $temp2$, $temp3 \leftarrow Compare(Align[1], [Align[2], Align[3], Align[4]])$
11: $min \leftarrow min(Align[1].getCorrespondance(), temp1, temp2, temp3)$
12: $AlignMin \cup min$
13: $temp2$, $temp3 \leftarrow Compare(Align[2], [Align[3], Align[4]])$
14: $min \leftarrow min(Align[2].getCorrespondance(), temp2, temp3)$
15: $AlignMin \cup min$
16: $temp3 \leftarrow Compare(Align[3], [Align[4]])$
17: $min \leftarrow min(Align[3].getCorrespondance(), temp3)$
18: $AlignMin \cup min$

L'algorithme 16 prend en entré un tableau **Align** qui contient l'alignement final des méthodes et retourne un alignement final selon la comparaison du min **AlignMin**.

- **Ligne 1-16** : Initialisation des variables.
- **Ligne 7-9** : Remplissage des variables temporaires en faisant la comparaison entre le premier alignement (Figure 2.13) qui contient le moins de correspondances avec les autres alignements finaux des méthodes. Calculer le minimum entre la similarité de la correspondance entre le premier alignement et les quatre variables temporaires. Ajouter la correspondance à l'alignement du minimum.
- **Ligne 10-12** : Remplissage des variables temporaires en faisant la comparaison entre le deuxième alignement avec les autres alignement finaux des méthodes. Calculer le minimum entre la similarité de la correspondance entre le deuxième alignement et les trois variables temporaires. Ajouter la correspondance à l'alignement du minimum.
- **Ligne 13-15** : Remplissage des variables temporaires en faisant la comparaison entre le troisième alignement avec les autres alignement finaux des méthodes. Calculer le minimum entre la similarité de la correspondance entre le troisième alignement et les deux variables temporaires. Ajouter la correspondance à l'alignement du minimum.
- **Ligne 16-18** : Remplissage de la variable temporaire en faisant la comparaison entre le quatrième alignement avec le cinquième alignement final. calculer

le minimum entre la similarité de la correspondance entre le quatrième alignement et la variable temporaire. Ajouter la correspondance à l'alignement du minimum.

La figure (2.14) montre un exemple détaillé de l'application de l'algorithme 16.

Cet algorithme va comparer le premier alignement avec les deux autres. On

Figure 2.14: Extrait du résultat d'application de l'algorithme d'agrégation et de comparaison avec le Min

commence par la première correspondances (Film, movies, 0.588) de jaccard, on la compare avec (Film, movies, 0.98) de paris ensuite avec (Film, movies, 0.6) de ProbaMap. On prend le minimum et on l'ajoute à alignMin. Puis, on efface la correspondance (Film,movies) des alignements à comparer, car elle existe déjà dans l'alignement minimum. Ensuite, nous procédons de la même manière avec la correspondances (musique, music, 0.888). Une fois que toutes les correspondances du premier alignement sont comparées avec les autres alignements, nous pourrons passer à la deuxième étape, c'est à dire, la comparaison du deuxième alignement avec le troisième qui sera la même que la précédente.

Le même principe est appliqué aux deux autres algorithmes de la comparaison et d'agrégation avec le Max et la Moyenne.

L'algorithme 17 combine les trois algorithmes précédents pour faire sortir les trois alignement finaux du min, max et la moyenne.

Algorithm 17 Agrégation et comparaison

Input: *Alig*: Tableau qui contient l'alignement de toutes les méthodes.
Output: *AlignFinal*: Tableau de alignement final contenant les trois alignements (Min, Max et la Moye

1: $AlignMin \leftarrow \emptyset$
2: $AlignMax \leftarrow \emptyset$
3: $AlignMoy \leftarrow \emptyset$
4: $AlignMin \leftarrow MinComp(Align)$
5: $AlignMax \leftarrow MaxComp(Align)$
6: $AlignMoy \leftarrow MoyComp(Align)$
7: $AlignFinal[] \leftarrow \emptyset$
8: $AlignFinal[0] \leftarrow AlignMin$
9: $AlignFinal[1] \leftarrow AlignMax$
10: $AlignFinal[2] \leftarrow AlignMoy$ **return** $AlignFinal$

L'algorithme 17 prend en entré un tableau contenant les alignements finaux des méthodes **Align** et retourne un tableau de l'alignement final contenant les trois alignements (Min, Max, Moyenne) **AlignFinal**.

- **Ligne 1-3**: Initialisation des variables.
- **Ligne 4-6**: Extraction des alignements finaux du Min, du Max et de la Moyenne.
- **Ligne 7-10**: Affectation des alignements finaux à un tableau et le retourné comme résultat.

2.3 Spécification des métriques pour l'évaluation du système AlignFX

Avant de présenter les trois algorithmes pour l'évaluation de notre système, nous allons d'abord définir les métriques utilisées pour évaluer les différentes méthodes de similarités et de probabilités. L'évaluation des méthodes s'est faite à base des trois métriques suivantes :

- **La précision**

  Elle correspond à la mesure du ratio des bonnes correspondances trouvées sur le total des correspondances retournées par la méthode choisie.

$$Precision = \frac{Correspondances\ correctes}{total\ des\ correspondances} \tag{2.3}$$

- **Le rappel** C'est la mesure du ratio des bonnes correspondances trouvées sur les correspondances attendues.

$$Rappel = \frac{Correspondances\ correctes}{correspondances\ attendues} \tag{2.4}$$

- **F-mesure** Même si les mesures citées ci dessus, sont des mesures fiables et très utilisées, dans certains cas il est recommandé d'utiliser une seule mesure qui est basée sur la précision et le rappel, il s'agit de la F-mesure.

$$F - mesure = \frac{2 \times rappel \times precision}{precision + rappel} \tag{2.5}$$

Dans ce qui va suivre, nous allons décrire brièvement les algorithmes d'évaluation réalisés. Nous allons commencer par présenter l'algorithme qui compare les correspondances obtenues des différentes méthodes avec celle de l'alignement initial de référence.

Algorithm 18 Dénombrement des correspondances correctes

Input: ref: Alignement de référence, arr: Alignement final d'une méthode.
Output: $Count$: Compteur du nombre de correspondances correctes.

1: $Count \leftarrow 0$
2: **for** $Cr \in arr$ **do**
3: **for** $Cr2 \in ref$ **do**
4: **if** $Cr.getleftPart() = Cr2.getleftPart()$ And $Cr.getrightPart() = Cr2.getrightPart()$ **then**
5: $Count++$
return $Count$

L'algorithme 18 prend en entré un alignement de référence (**ref**), et un alignement final d'une méthode (**arr**), et vérifie si la partie gauche et la partie droite des deux correspondances sont égales. Il retourne comme résultat (**Count**) qui donne le nombre de correspondances correctes que l'algorithme à détecter.

L'algorithme 19 prend en entré un alignement de référence et un alignement final d'une méthode, et retourne un tableau **évaluation** qui contient le calcul des trois mesures à savoir la précision, le rappel et la F-mesure.

Algorithm 19 Evaluation des alignements finaux

Input: ref: L'alignement de référence, arr: L'alignement final d'une méthode.
Output: $evaluation$: Tableau d'évaluation qui contient la précision, le rappel, la F-mesure

1: $precision \leftarrow 0$
2: $recall \leftarrow 0$
3: $fMeasure \leftarrow 0$
4: $eval \leftarrow evaluate(ref, arr)$
5: $evalaluation \leftarrow \emptyset$
6: $found \leftarrow arr.size()$
7: $correct \leftarrow eval[0]$
8: $total \leftarrow ref.size()$;
9: $eval \leftarrow \emptyset$
10: $precision \leftarrow correct/found$
11: $recall \leftarrow correct/total$
12: $fMeasure \leftarrow 2 * precision * recall/(precision + recall)$
13: $evalaluation[0] \leftarrow precision$
14: $evalaluation[1] \leftarrow recall$
15: $evalaluation[2] \leftarrow fMesure$
return $evaluation$

L'algorithme 20 compare les différentes mesures de l'évaluation obtenues de l'algorithme 19, c'est à dire, les évaluations des alignements min, max et moyenne (**evaluationAll**) et retourne un tableau **évaluation** proposant la meilleure évaluation.

Algorithm 20 Comparaison des évaluations

Input: $evaluationAll$: Tableau d'évaluation qui contient l'évaluation du min, max et de la moy
Output: $evaluation$: La meilleur évaluation

1: $Max \leftarrow Max(evaluationAll.getMinEval().getfmesure(),$
$Max(evaluationAll.getMaxEval().getfmesure(), evaluationAll.getMoyEval().getfmesure()))$
2: **if** $evaluationAll.getMinEval().getfmesure() == Max$ **then**
3: **return** $evaluationAll.getMinEval()$
4: **else if** $evaluationAll.getMaxEval().getfmesure() == Max$ **then**
5: **return** $evaluationAll.getMaxEval()$
6: **else if** $evaluationAll.getMoyEval().getfmesure() == Max$ **then**
7: **return** $evaluationAll.getMoyEval()$

2.4 Conclusion

Dans ce chapitre, nous avons présenté l'architecture globale de notre système qui reflète le raisonnement utilisé pour résoudre le problème de l'alignement sémantique entre deux ontologies, en appliquant les différents algorithmes de similarité et de probabilité. Nous avons aussi présenté le processus d'agrégation qui permet de donner en sortie le meilleur alignement final entre les comparaisons du min, du max et de la moyenne.

Implementation

Chapitre 3

Implémentation de AlignFX

3.1 Introduction

Après avoir décrit les différentes techniques de conception dans le chapitre précédent, ainsi que les différents modules qui composent notre système, nous présenterons dans ce chapitre l'environnement matériel et logiciel dans lequel ce travail a été implémenté ainsi que la description de la plateforme *AligFX* à travers l'ensemble de ses interfaces.

3.2 Environnement de travail

3.2.1 Environnement matériel

la plateforme *AlignFX* a été implémentée sur une machine *ASUS* dont la configuration est décrite dans Table 3.1 :

Processeur	Intel Core i7-7500U 2.70Ghz 2.90Ghz
Mémoire (RAM)	6Go
Système d'exploitation	Windows 10 x64

Table 3.1: Configuration de la machine de développement

3.2.2 Logiciels et Frameworks utilisés

IntelliJ est un environnement de développement intégré (IDE) utilisé dans la programmation informatique, en particulier pour le langage Java, mais aussi Kotlin et plein d'autre. Il est développé par la société *JetBrains* et fournit une analyse de code, un débogueur graphique, un testeur unitaire intégré, une intégration avec les systèmes de contrôle de version (VCSes).

JavaFX est une plate-forme d'application client open source de nouvelle génération pour les systèmes de bureau, mobiles et embarqués basés sur Java. Il s'agit d'un effort de collaboration entre de nombreuses personnes et entreprises dans le but de produire une boîte à outils moderne, efficace et complète pour le développement d'applications client riches.

GraphStream est une bibliothèque Java de gestion de graphes qui se concentre sur les aspects dynamiques des graphiques. Il se concentre principalement sur la modélisation de réseaux d'interaction dynamique de différentes tailles.
L'objectif de la bibliothèque est de fournir un moyen de représenter des graphiques et de travailler dessus. À cette fin, GraphStream propose plusieurs classes de graphes qui permettent de modéliser des graphes dirigés et non dirigés, des graphes 1 ou des p-graphes (alias multigraphes, qui sont des graphes pouvant avoir plusieurs arêtes entre deux nœuds). GraphStream permet de stocker tout type d'attribut de données sur les éléments du graphe : nombres, chaînes ou n'importe quel objet.
De plus, en outre, GraphStream fournit un moyen de gérer l'évolution du graphique dans le temps. Cela signifie gérer la façon dont les nœuds et les arêtes sont ajoutés et supprimés, ainsi que la façon dont les attributs de données peuvent apparaître, disparaître et évoluer.

Scene Builder est un outil open source qui permet la conception par drag and drop pour la créations d'interfaces utilisateur JavaFX, la séparation des fichiers de conception et de logique permet aux membres de l'équipe de se concentrer rapidement et facilement sur leur couche spécifique de développement d'applications.

Figma est un éditeur de graphiques vectoriels et un outil de prototypage. Il est principalement basé sur le web, avec des fonctionnalités hors ligne supplémentaires activées par des applications de bureau pour macOS et Windows. Les Figma Mirror companion apps pour Android et iOS permettent de visualiser des prototypes Figma sur des appareils mobiles. L'ensemble des fonctionnalités de Figma est axé sur l'utilisation dans la conception de l'interface utilisateur et de l'expérience utilisateur, en mettant l'accent sur la collaboration en temps réel.

Protégé est un système auteur pour la création d'ontologies. Il a été créé à l'université Stanford et est très populaire dans le domaine du Web sémantique et au niveau de la recherche en informatique. Protégé est développé en Java. Il est gratuit et son code source est publié sous une licence libre (la Mozilla Public License).
Protégé peut lire et sauvegarder des ontologies dans la plupart des formats d'ontologies : RDF, RDFS, OWL, etc.

3.2.3 Langages utilisés

Java est un langage de programmation orienté objet et une plate-forme informatique qui ont été créés par Sun Microsystems en 1995. Beaucoup d'applications et de sites Web ne fonctionnent pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable. Des ordinateurs portables aux centres de données, des consoles de jeux aux superordinateurs scientifiques, des téléphones portables à Internet, la technologie Java est présente sur tous les fronts .

3.3 Remplissage des ontologies

Dans ce travail nous avons utilisé deux ontologies, une en français et l'autre en anglais. Le remplissage des ontologies a été fait en deux parties :

- **Extraction des instances à partir de Yago et de DBpedia** avec les requêtes sparql écrites dans l'éditeur de requête sparql fournit par Yago et DBpedia. Des exemples de requêtes sont illustrés dans les figures (3.1) et (3.2)

Figure 3.1: Exemple de requête sparql dans Yago

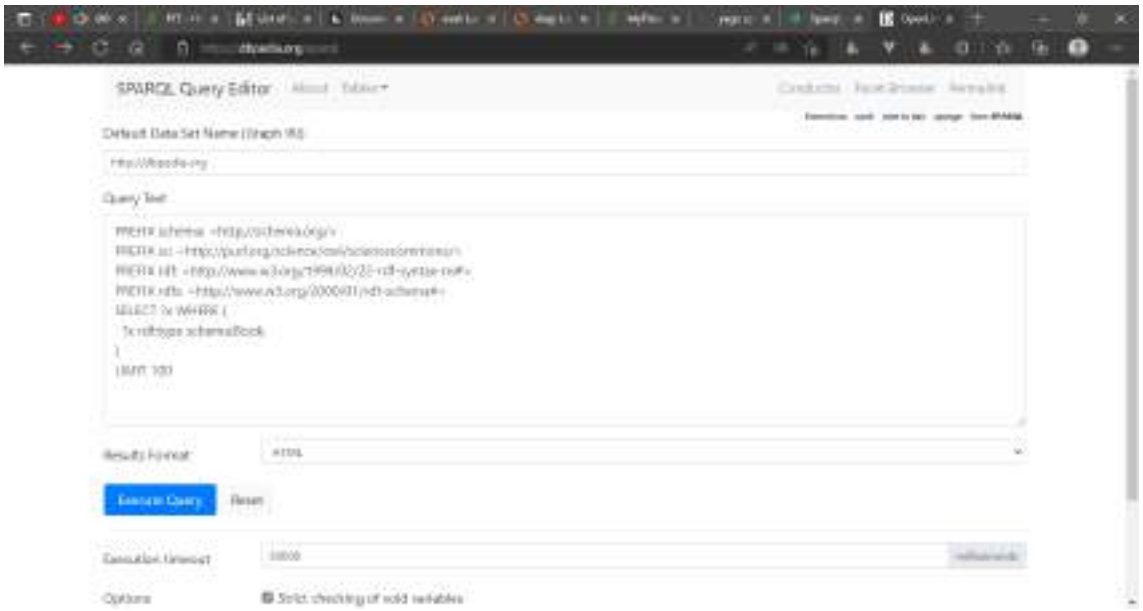


Figure 3.2: Exemple de requête sparql dans Dbpedia

Une fois les requêtes executées, nous pouvons télécharger les résultats sous format RDF/XML (Figure 3.3)

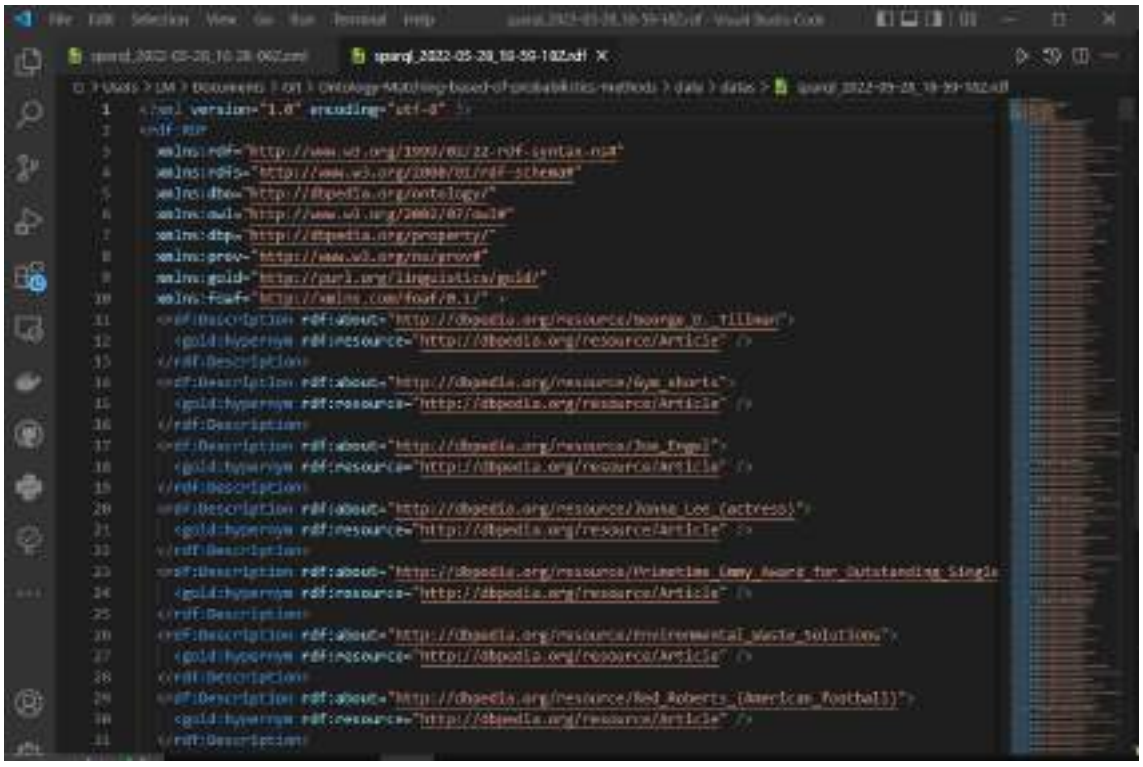


Figure 3.3: Exemple de résultat d'une requête SPARQL sous format RDF/XML

Dans la figure on peut voir dans la balise *<rdf:Description>*, l'attribut *rdf:about* qui contient l'instance que nous cherchons, et dans la balise *<gold:hypernym>*, l'attribut *rdf:resource* contient le concept associé à cette instance.

- **Remplir l'ontologie** : Ensuite vient l'étape de remplissage des ontologies (2.2.1) en utilisant l'algorithme de remplissage des ontologies vu dans la même section.

  Par faute de performance matériel, nous avons rempli nos deux ontologies de *dix concepts*, chaque concept possède entre cent et trois-cent instances.

Table 3.2: Informations supplémentaire sur les ontologies

Ontologies	***# Nombre de concepts***	***#nombre d'instances***	***Langue***
O1	10	2715	Français
O2	10	2547	Anglais

3.4 Arborescence de notre application

Notre application se divise en plusieurs modules qui peuvent être vu dans la figure (3.4)

Figure 3.4: La structure du répertoire de AlignFX dans l'ide intelliJ

- *AlignFX/app/main.java* : C'est le point de départ de l'application. Ce fichier est responsable du lancement de l'application.

- *AlignFX/resource* : Ici ce trouve la partie design de l'application, en effet à ce niveau nous trouvons tout les fichiers **.fxml** qui sont responsables du design de l'application.

- *AlignFX/controllers* : Ici ce trouve les controllers, c'est la partie logique des interfaces utilisées dans l'application. *AlignFX/model* : définit les classes de notre application.

- *AlignFX/MachingMethods* : contient les fonctions qui vont servir à calculer les méthodes de similarités et des méthodes probabilistes vues dans la 2.2.4.

- *AlignFX/util* : contient toutes les autres fonctionnalités. Par exemple, les fonctions qui servent à l'extraction des concepts et des instances(2.2.2) ou de nettoyage (2.2.3) etc.

3.5 Présentation de la plate-forme AlignFX

Dans cette section, nous présenterons la plateforme AlignFX et les différentes interfaces qui la constituent.

3.5.1 Écran d'accueil

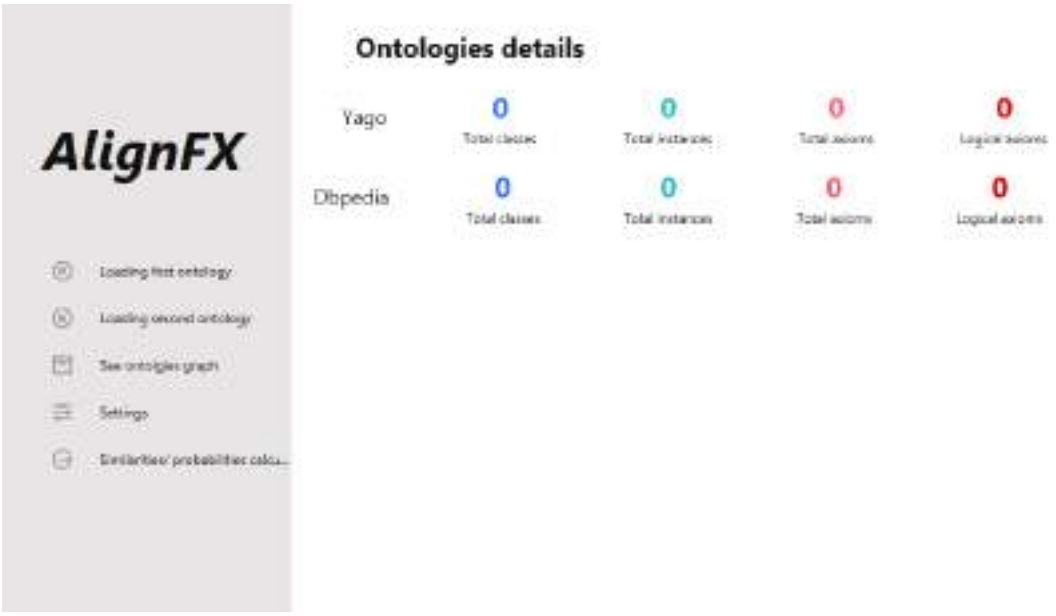


Figure 3.5: Écran d'accueil.

Le lancement de l'application se fait par l'affichage de la première fenêtre *mainTest.fxml* qui sera visualisée dans une fenêtre d'application. Cette interface constitue la porte d'accès à l'application. La Figure (3.5) représente la page d'accueil, en cliquant sur *Load first ontology*, il nous sera possible de choisir la première ontologie, même chose pour *Load second ontology*. Une fois les deux ontologies choisies, les informations sur l'ontologie devrait s'afficher (Figure 3.6).

Comme vous le voyez (Figure (3.6) l'utilisateur pourra visualiser les détails, à savoir, le nombre de concepts, des instances etc..

Figure 3.6: Écran d'accueil avec les informations sur les ontologies.

Affichage de l'ontologie

L'utilisateur peut visualiser un graphe des ontologies données s'il appuie sur *See ontologie graph*, il accédera à deux fenêtres, chacune montrant un graphe d'une ontologie (Figure 3.7), dans laquelle l'utilisateur peut voir les ontologies de manière

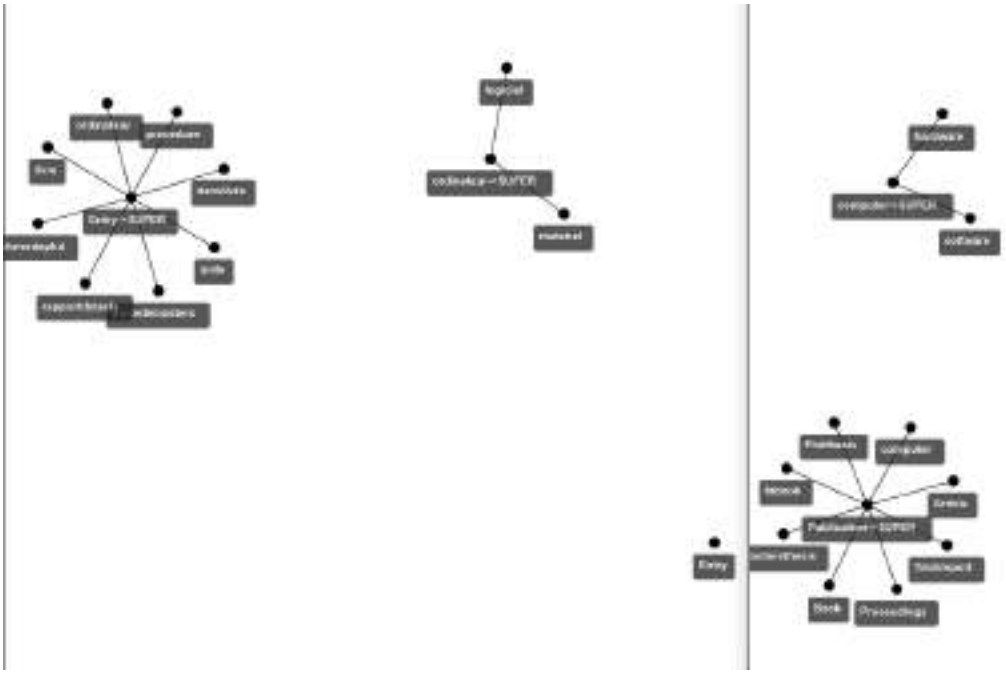


Figure 3.7: Graphes des deux ontologies

claire est précise. Les noeuds marqués par *->SUPER* sont des super classes et les autres sont des sous classes. Par exemple, *ordinateur->SUPER* est une super classe et ses sous classes sont matériel et logiciel. Maintenant, si l'utilisateur actionne le bouton *Similarities/probabilities calculation*, il sera redirigé vers le prochain écran qui est celui du calcul de similarité et probabilité.

3.5.2 Écran de calcul et de comparaison

Ici l'utilisateur est invité à choisir entre une ou toutes les méthodes pour calculer les similarité/probabilités, pour ce faire, il devra appuyer sur le bouton *calculate similarities/ probabilities* pour afficher les différents onglets des méthodes (allant de la méthode levenshtein jusqu'à la méthode probaMap).

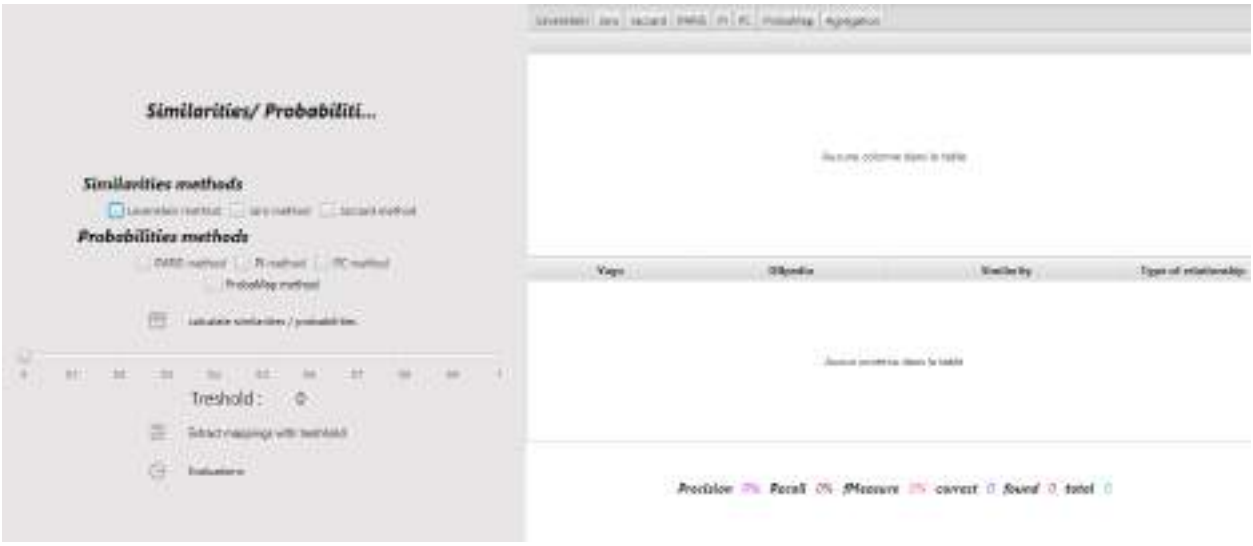


Figure 3.8: Affichage des méthodes de similarités supportées par le système

Processus de calcul de similarité/probabilité

Une fois que l'utilisateur sélectionne les méthodes avec les *checkbox* associés, il peut appuyer sur le bouton *calculate similarities/probabilities* pour calculer les similarité/probabilité entre les méthodes (2.2.4). La figure (3.9) montre le résultat de l'opération sur l'onglet de la méthode PARIS.

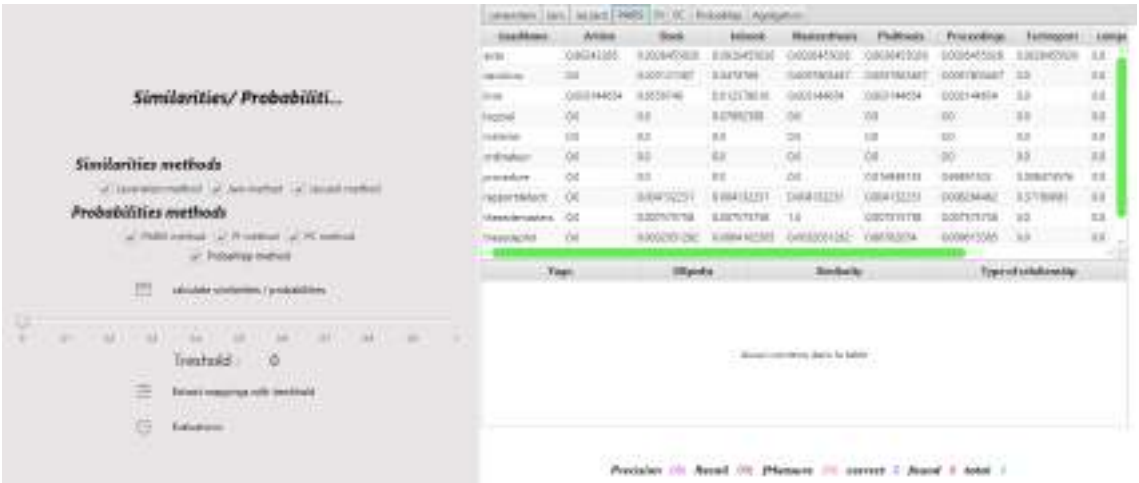


Figure 3.9: Resultat de calcul et de comparaison du processus de similarité/probabilité

Processus d'extraction des correspondances

Une fois les similarités/probabilités calculées, l'utilisateur aura le choix de choisir un *seuil* pour extraire des correspondances pertinentes, ce choix se fait en ajustant le *slider* illustré dans la figure (3.9) qui lui permettra d'extraire les correspondances pertinentes 2.2.5. Une fois le seuil choisi, l'utilisateur est invité à appuyer sur le bouton *Extract mapping with treshold*. Une fois le bouton appuyé trois opération sont déclenchées :

- D'abord le tableau ce verra rempli avec les correspondances qui ont une similarité/ probabilité au dessus du seuil.
- Chaque méthode ce verra évaluée sur ses correspondances extraites.
- Enfin le processus d'agrégation (2.2.6) ce lance et les résultats seront disponibles dans l'onglet Agrégation.

Figure 3.10: Écran de calcul/comparaison avec les correspondances extraites et leurs évaluations

La figure (3.10) montre un exemple d'affichage après que l'utilisateur ait défini un seuil de 0,5. Cette fenêtre visualise les valeurs de correspondances qui ont une similarité/probabilité en dessus du seuil après l'extraction de l'alignement final de chaque méthode. Il y a aussi l'affichage de l'évaluation de la méthode avec les mesures précision, rappel et F-Mesure.
Maintenant que les correspondances pertinentes ont été extraites, l'utilisateur peut accéder à l'onglet Agrégation afin de combiner différents résultats trouvés pour extraire le meilleur alignement.

Processus d'agrégation

Le processus d'agrégation utilise les correspondances extraites de toutes les méthodes et génère trois alignements selon la comparaison du min, du max et de la moyenne. Les trois alignements seront visualisés et après leurs évaluation, il y a affichage du meilleur alignement. La figure (3.11) montre le résultat de cette opération. Nous avons l'affichage de l'alignement de la moyenne, qui est considéré comme le meilleur

Figure 3.11: Écran de calcul/comparaison - onglet Agrégation

alignement, car il présente une meilleure F-mesure.
A ce stade, l'utilisateur peut appuyer sur le bouton *Évaluations*, il y a ouverture d'une fenêtre, où nous pouvons comparer les méthodes individuellement avec des graphes etc...

3.5.3 Écran Evaluation détails

Figure 3.12: Onglet Main : Détails évaluation

La figure (3.12), présente une interface relative à *Evaluation Détails* qui propose deux onglets, *Main* et *Evaltab* :

- *L'onglet Main* affiche le récapitulatif des performances des méthodes sous forme de tableau.
- *L'onglet evaltab* affiche deux graphes, l'un montrant la précision des méthodes avec différents seuils utilisés pour extraire les correspondances.

Onglet Main

L'utilisateur est invité à appuyer sur le bouton *Setup Evaluation Table*, qui affiche (Figure 3.13) un résumé des évaluations faites dans l'interface précédente.

Figure 3.13: Onglet Main : Détails Évaluation : Affichage des performances des méthodes.

Onglet EvalTab

L'utilisateur est invité à actionner le bouton *See evaluation graph* qui affichera les graphes d'évaluations (figure fig(3.14).

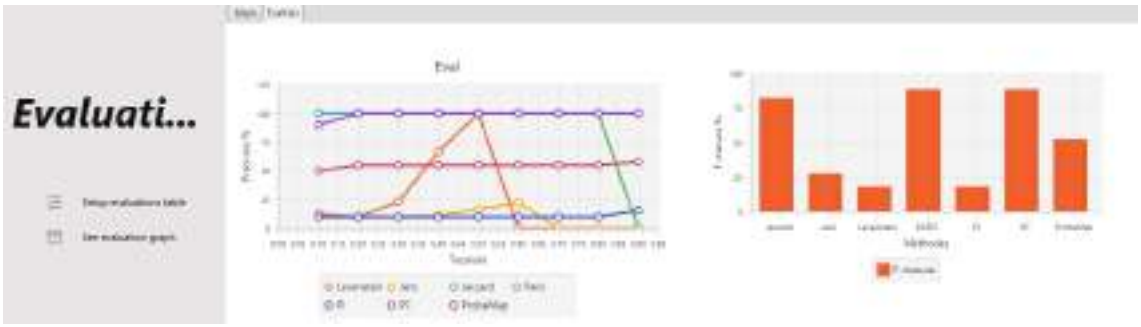


Figure 3.14: Onglet EvalTab : affichage des graphes d'evaluation

3.6 Expérimentations et Résultats

Dans cette section, nous allons décrire les expériences menées sur nos deux ontologies O1 et O2 pour prouver que les méthodes probabilistes donnent de bien meilleurs résultats que les méthodes de similarités. Pour atteindre ce but, nous avons abordé deux parties. Dans la première partie, nous verrons les correspondances extraites de toutes les méthodes similarités/probabilistes en utilisant un seuil fixe. Dans la deuxième partie, nous discuterons les performances des méthodes, d'abord en montrant une évaluation de leurs performances en utilisant la f-mesure, et ensuite en montrant l'évolution de leurs précisions après variation du seuil.

3.6.1 Dispositif Expérimental

Pour l'expérimentation, nous avons utilisé nos deux ontologies O1 et O2, où chaque ontologie renferme dix concepts, l'ontologie O1 possède 2715 instances et l'ontologie O2 possède 2545 instances. L'ontologie O1 est en français et O2 est en anglais.
Pour évaluer les méthodes de similarités/probabilistes 2.3 nous avons utilisé la Précision, le Rappel et la F-mesure.

3.6.2 Discussion des résultats

Le tableau 3.3 donne le résultat des expérimentations avec un seuil de 0.5 appliqué à toutes les méthodes.

Table 3.3: Résultat après extraction des correspondances pertinentes de chaque méthode

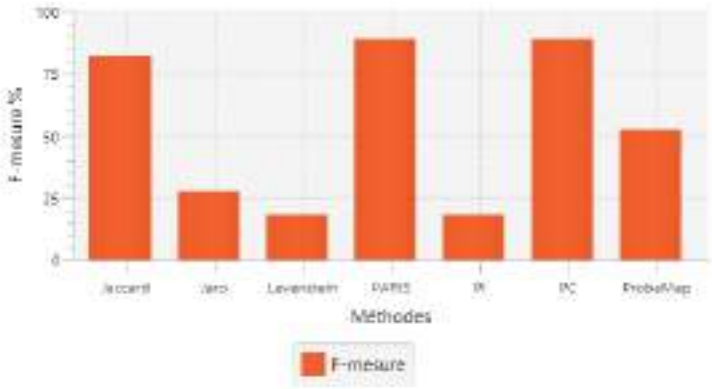
Méthode	***# Seuil***	***#correspondances trouvées***
Levenshtein	0.5	1
Jaro	0.5	48
Jaccard	0.5	7
Paris	0.5	8
PC	0.5	8
PI	0.5	100
ProbaMap	0.5	28

Nous avons choisi un seuil neutre qui nous permettra d'extraire les correspondances pertinentes. Ainsi, nous remarquons que la méthode Levenshtein a extrait qu'une seule correspondance, ce qui prouve l'inconvénient des méthodes basées sur la comparaison entre chaînes de caractères, c'est à dire, entre concepts quand les ontologies sont de langues différentes.

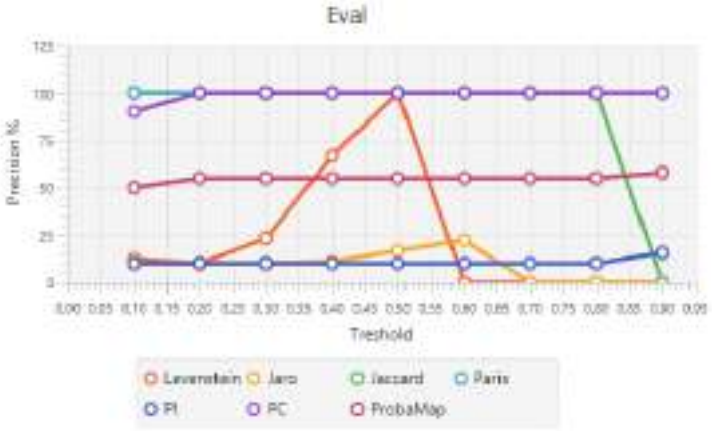
La figure 3.15**a** montre la f-mesure de chaque méthode pour l' extraction des correspondances avec un seuil de 0.5. Nous pouvons voir à partir du graphe, que les méthodes probabilistes dominent les méthodes de similarité, sauf pour la méthode de jaccard qui a une bonne F-mesure, mais comme ses correspondances sont extraites des relations d'équivalences, il y a une perte importante de correspondances.

La figure 3.15**b** montre l'évolution de la précision des méthodes avec la variation du seuil pour l'extraction des correspondances.

Nous pouvons constater que les méthodes PC, PARIS et Jaccard sont à cent pourcent de précision à partir du seuil 0.2. Après le seuil 0.8, la méthode Jaccard ne donne plus de bonnes correspondances, alors que les méthodes PARIS et PC sont toujours à cent pourcent de précision. Ces résultats prouvent que dans le cas où les ontologies à aligner sont de différentes langues, l'utilisation des méthodes probabilistes est beaucoup plus appropriée.



(a) Résultat de l'application de la F-mesure sur toutes les méthodes avec un seul de 0.5



(b) Résultat de l'application de la précision sur les méthode avec la variation du seuil.

Figure 3.15: Résultats Évaluation

3.7 Conclusion

Dans ce chapitre, nous avons décrit les différents modules composant notre système *AlignFX*, en détaillant leurs implémentations et en mettant en évidence les fonctionnalités de chaque module. Puis, nous avons présenté une vue globale sur la façon dont nous avons évalué les méthodes de similarités/probabilités, la méthode d'extraction des correspondances, le processus d'agrégation, et en dernier, nous avons détaillé nos études expérimentales qui nous ont permis de prouver que les méthodes probabilistes sont meilleures et plus performantes dans le cas d'une différences de langue entre les ontologies.

Conclusion et Perspectives

Dans ce travail, nous avons mis en évidence l'intérêt de l'utilisation des méthodes probabilistes par rapport aux méthodes de similarité.

L'objectif de notre travail est de prouver que les méthodes probabilistes sont plus performantes que les méthodes de similarités. Pour atteindre cet objectif, nous avons présenté dans le premier chapitre, les concepts de base de l'alignement sémantique et présenté quelques méthodes implémentées et utilisées dans nos expériences.

Ensuite, au niveau du chapitre deux, nous avons présenté l'architecture globale de notre système, **AlignFX**, en détaillant le processus d'alignement sémantique à travers les différents modules et leurs fonctionnalités respectives.

Enfin, au niveau du chapitre trois, nous avons décrit l'implémentation de notre système, **AlignFX**, en présentant chaque interface avec leurs fonctionnalités, et les résultats de nos expériences. Ces dernières ont montré la performance des méthodes probabilistes par rapport aux méthodes de similarité.

Parmi les perspectives relatives aux quelques extensions futures de notre travail, plusieurs points nous semblent dignes d'un intérêt particulier :

- Intensifier l'expérimentation en considérant d'autres ontologies.
- Envisager d'autres types d'alignement tels que l'alignement entre les instances.
- Implémenter d'autres méthodes probabilistes.

Bibliography

[1] M. Ehrig. "Ontology Alignment : Bridging the Semantic Gap, Semantic Web And Beyond Computing for Human Experience 4". In: *SPRINGER* (2007), pp. 1–250.

[2] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching, Second Edition*. Springer, 2013. ISBN: 978-3-642-38720-3.

[3] Naima El Ghandour, Moussa Benaissa, and Yahia Lebbah. "An Integer Linear Programming-Based Method for the Extraction of Ontology Alignment". In: *Int. J. Inf. Technol. Web Eng.* 16.2 (2021), pp. 25–44. DOI: 10.4018/IJITWE.2021040102. URL: https://doi.org/10.4018/IJITWE.2021040102.

[4] Thi Thuy Anh Nguyen and Stefan Conrad. "Ontology Matching using Multiple Similarity Measures". In: *KDIR 2015 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015), Volume 1, Lisbon, Portugal, November 12-14, 2015*. Ed. by Ana L. N. Fred et al. SciTePress, 2015, pp. 603–611. DOI: 10.5220/0005615606030611. URL: https://doi.org/10.5220/0005615606030611.

[5] Thomas R.Gruber. "translation approach to portable ontology specifications". In: 1993, pp. 199–220. DOI: 10.1006/knac.1993.1008.

[6] Joerg Schoenfisch et al. "Root cause analysis in IT infrastructures using ontologies and abduction in Markov Logic Networks". In: *Inf. Syst.* 74.Part (2018), pp. 103–116. DOI: 10.1016/j.is.2017.11.003. URL: https://doi.org/10.1016/j.is.2017.11.003.

[7] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. "Ontology Alignment at the Instance and Schema Level". In: *CoRR* abs/1105.5516 (2011). arXiv: 1105.5516. URL: http://arxiv.org/abs/1105.5516.

[8] Rémi Tournaire et al. "Discovery of Probabilistic Mappings between Taxonomies: Principles and Experiments". In: *J. Data Semant.* 15 (2011), pp. 66–101. DOI: 10.1007/978-3-642-22630-4_3. URL: https://doi.org/10.1007/978-3-642-22630-4%5C_3.

[9] Mike Uschold. "Knowledge level modelling: concepts and terminology". In: *Knowl. Eng. Rev.* 13.1 (1998), pp. 5–29. URL: http://journals.cambridge.org/action/displayAbstract?aid=35973.