Approches vers des modèles unifiés pour l'intégration de bases de connaissances

Maria Koutraki

NNT : 2016SACLV082

THESE DE DOCTORAT
DE
L'UNIVERSITE PARIS-SACLAY
PREPAREE A
UNIVERSITE DE VERSAILLES-SAINT-QUENTIN-EN-YVELINES

ECOLE DOCTORALE N°580
Sciences et technologies de l'information et de la communication

Spécialité de doctorat : Informatique

Par

Mme Maria Koutraki

Approches vers des modèles unifiés pour l'intégration de bases de connaissances

Thèse présentée et soutenue à Versailles, le 27ème septembre 2016 :

Composition du Jury :

M. Amann Bernd,	Professeur, Université Pierre & Marie Curie,	Président du Jury
M. Vodislav Dan,	Professeur, Université Cergy-Pontoise,	Co-directeur de thèse
Mme Preda Nicoleta,	Maître de Conférences, UVSQ,	Co-encadrant de thèse
Mme Zeitouni Karine,	Professeure, UVSQ,	Directrice de thèse
M. Christophides Vassilis,	Professeur, Université de Crète,	Rapporteur
Mme Rousset Marie-Christine,	Professeure, Université de Grenoble,	Rapporteur
M. Goasdoué François,	Professeur, Université de Rennes 1,	Examinateur
Mme Saïs Fatiha,	Maître de Conférences, Université Paris-Sud,	Examinatrice

To my family, for their eternal love and support ...

Titre : Approches vers des modèles unifiés pour l'intégration de bases de connaissances

Mots clés : intégration de données, Web sémantique, service Web, alignement de relations dans les ontologies

Résumé : L'avènement du World Wide Web, en particulier le Web 2.0, a conduit à des changements drastiques dans la façon dont l'information est créée et consommée sur le Web. À l'heure actuelle, il y a plus de 3 milliards d'utilisateurs sur le Web, qui, de façon explicite ou implicite, contribuent à la création de contenu. Ce contenu reflète la dynamique et l'hétérogénéité des utilisateurs et des applications Web utilisées pour le produire (par exemple des réseaux sociaux, des blogs, des pages Web, etc.). Les estimations indiquent que le nombre de pages est de l'ordre des trillions. Le contenu résultant prend la forme d'ensembles de données disparates, dont les principales caractéristiques sont: (i) le volume de données, (ii) le domaine représenté, (iii) les schémas et autres moyens de représenter le contenu, et (iv) la dynamique et la nature évolutive des données.

Différents domaines de recherche se sont intéressés à l'exploitation de la richesse du contenu du Web 2.0, qui constitue une tâche non triviale. Le Web sémantique, avec sa vision du paysage Web où les données sont uniformément interprétables et partagées entre les applications, est l'une de directions de recherche majeures. Pourtant, depuis sa création, il reste encore un grand nombre de défis sur la façon d'intégrer les différentes sources d'information dans un paysage de données unifié. Des initiatives récentes, comme le Linked Open Data, comptent des milliers de jeux de données contenant plus de 30 milliards de faits, exprimés sous forme de données structurées suivant le modèle du Web sémantique. Toutefois, la manière décentralisée de publier ces données soulève d'autres problèmes tels que la diversité et la redondance des schémas utilisés pour les représenter, menant à une grande hétérogénéité de ces ensembles de données au niveau du schéma.

De plus, dans de nombreux cas, les données sont accessibles à la demande, à travers des interfaces de programmation (API) tels que les services Web, qui codent des opérations complexes exécutées sur des sources de données spécifiques. Il existe aujourd'hui des dizaines de milliers de services Web, leur nombre étant en augmentation continue, chaque service fournissant une multitude d'opérations avec des caractéristiques spécifiques en termes de paramètres d'entrée / sortie. L'accès aux fonctionnalités et aux données fournies à travers ces services Web représente un challenge à cause de l'hétérogénéité des formats de sortie et des représentations utilisées par les différents services.

Dans cette thèse, nous identifions plusieurs défis liés au Web sémantique (Linked Data) et aux services Web, plus précisément à la difficulté d'utilisation de toute la richesse du Web 2.0, notamment les sources de données Linked Data et celles exposées par le biais de services Web. Par conséquent, nous adressons deux problèmes principaux, qui correspondent aux contributions de cette thèse: (i) *DORIS*, une méthode de traduction automatique des résultats d'appels de services Web, et (ii) *SOFYA*, une approche pour l'alignement automatique de relations dans les ontologies.

DORIS. Dans la première partie de cette thèse, nous abordons le problème de l' intégration de services Web, à savoir l'intégration des résultats d'appels d'opérations de services Web par le biais d'un schéma global. A cet effet, nous proposons des approches qui infèrent automatiquement le schéma des résultats d'appels de services, en exploitant

l'intersection de ces résultats d'appels avec les instances de bases de connaissances du monde réel, telles que DBpedia ou YAGO.
Plus précisément, nous modélisons une opération de service Web comme une *view with binding patterns* sur le schéma global de la base de connaissances. A cet effet, nous présentons des approches automatiques qui infèrent la définition de cette vue sur le schéma global. Ensuite, nous montrons comment produire les fonctions de transformation des résultats d'appels de service dans ce schéma. Nos expérimentations sur plus de 50 services Web du monde réel montrent que nous pouvons automatiquement déduire le schéma avec une précision de 81% -100%.
SOFYA. Dans la deuxième partie de cette thèse, nous abordons le problème de l' alignement de relations dans les ontologies Linked Data. Etant donnée une relation dans l'ontologie d'une source de données Linked Data, nous trouvons les relations de l'ontologie d'une autre source de données cible, qui subsument notre relation de départ. De plus, étant donnés les problèmes de passage à l'échelle en nombre de sources et volume de données, ainsi que d'évolution constante du contenu de ces sources, nous avons considéré l'alignement de relations dans un cadre online, où l'accès aux sources se fait par des appels de services à travers des points d'accès SPARQL. Pour assurer l'efficacité de notre approche, nous procédons à l'alignement en extrayant de petits échantillons à partir des sources de données considérées. L'approche *SOFYA* se base sur un modèle d'apprentissage supervisé, utilisant un ensemble de caractéristiques d'alignement qui inclut des règles d'association en termes d'instances des relations candidates, la similitude lexicale des noms de relation, ainsi que et d'autres statistiques sur les instances des relations. Nous effectuons une évaluation approfondie de notre approche sur trois bases de connaissances, DBpedia, YAGO et Freebase. Nous montrons que nous pouvons effectuer l'alignement avec une précision moyenne de 82% et un maximum de 100%. En outre, l'approche peut être appliquée en temps réel, puisque la surcharge introduite en termes de temps d'exécution et de bande passante est minime.
En conclusion, les contributions de cette thèse peuvent être utilisées pour intégrer des services Web et interroger d'une manière uniforme un grand nombre de sources Linked Data. Ceci permet une intégration transparente des deux principaux types de sources de données structurées dans le Web 2.0.

Title : Approaches Towards Unified Models for Integrating Web Knowledge Bases

Keywords : data integration, semantic Web, Web services, ontology alignment

Abstract : The advent of World Wide Web, especially Web 2.0, has lead to drastic changes on how information is created and consumed on the Web. Currently, there are more than 3 billion Web users, who either through explicit or implicit means contribute to content creation. The content reflects the dynamics and heterogeneity of the users and Web applications used to generate such content (e.g. social networks, blogs, Web pages etc.). With estimates showing the number of pages being in trillions. The resulting content resides in disparate datasets, and some of the key features are: (i) scale of the data, (ii) domains, (iii) schemas and means of representing content, and (iv) dynamics and evolving nature of the data.

To harness the wealth of content in Web 2.0 is non-trivial task, and such it has gathered attention from different research fields. Semantic Web is one of the research directions, with its core vision of Web landscape where data is uniformly interpretable and shared across applications boundaries. Yet, since its invention, there are still a large number of

open challenges on how to integrate the different sources of information into a unified data landscape. Recent initiatives, like the Linked Open Data, count more than thousands of datasets with more than 30 billions of statements, that are expressed as structured data following the Semantic Web vision. However, considering the decentralized way of publishing linked data, additional problems are encountered such as diversity and redundancy on schemas used to represent such datasets, hence, leaving a disintegrated landscape of datasets at schema level.
Furthermore, in many cases, data is accessible on demand through application programmable interfaces API, such as Web services. Web services, encode complex operations that are executed over specific data sources. Similarly, in the case of Web services, their number is in the tens of thousands and ever increasing, and with each service providing a multitude of operations, who have specific requirements in terms of input/output parameters. Accessing the functionalities and data exposed through such Web services is in particular challenging due to the heterogeneity of the output formats and representations used from the different services.
In this thesis, we identify several challenges in terms of *linked data* and *Web services*, which hinder end-users to make use of the full wealth of Web 2.0, namely data sources expressed as linked data, and those exposed through Web services. Consequentially, we consider two main problems which we address, thus, resulting as contributions of this thesis: (i) *DORIS* a framework for mapping output from Web service operation calls, and (ii) *SOFYA* an approach for ontological relation alignment.

DORIS. In the first part of this thesis, we address the problem of Web service integration, namely integrating output from Web service operation calls by means of global schema. For this purpose, we propose approaches that infer the schema of the output from
Web service operation call by exploiting the *intersection* of the output with instances from real-world knowledge bases like DBpedia or YAGO. Specifically, we model a Web service operation as a *view with binding patters* over the global schema. For this purpose, we present automated approaches which infer the view definition of an operation into the global schema. Next, we show how to compute *transformation functions* that can transform API call results into this schema. Our experiments on more than 50 real-world Web services show that we can automatically infer the schema with a precision of 81%-100%.
SOFYA. In the second part of this thesis, we address the problem of ontological relation alignment for linked datasets. Given a relation from a source dataset, we find relations in a target dataset that *subsume* our starting relation. Furthermore, accounting for the scalability of such a process e.g. large number of linked datasets, and the evolving nature datasets, we set up the relation alignment in an *online setting*. To ensure the *efficiency* of our approach, we perform the alignment by taking small samples from the datasets under consideration. The approach in *SOFYA* corresponds to a supervised machine learning model, which we learn on a set of features that consider *association rules* in terms of overlapping statements between relation alignment candidate pairs, lexical similarity of relation names, and other statistics we gather from the instances assigned to the specific relations. We perform an extensive evaluation of our approach on three real-world knowledge bases, DBpedia, YAGO and Freebase. We show that we can perform the alignment with high accuracy with an average precision of 82% and a maximum of 100%. Furthermore, the approach can be applied during query execution as the introduced overhead in terms of time and network bandwidth is minimal.
Finally, the contributions of this thesis, can be used to integrate Web services and query in an uniform manner the large number of linked datasets. Thus, allowing for seamless integration of two major data sources in Web 2.0.

Acknowledgments

First of all, I would like to express my sincere thanks to my advisors, Prof. Dan Vodislav, Associate Prof. Nicoleta Preda and Prof Karine Zeitouni for their support throughout the years of my PhD studies. Especially to Dan and Nicoleta, their enthusiasm and desire for perfection was an inspiration for me. This thesis would not be possible without their insistence, help and guidance.

I am grateful to Marie-Christine Rousset and Vassilis Christophides for thoroughly reading my thesis and for their valuable feedback. Further, I would like to thank Bernd Amann, François Goasdoué, and Fatiha Saïs for being part of my examining committee; I am very honored.

I would also like to give many thanks to all my colleagues and friends in DAVID Lab and in ETIS Lab for their support and help. To Dimitris, Claudia, Hanane, Tici, Dai-Hai, Raef, Ali, Twiza, Kenza, Mariam, Amelie, Abdul .. Thank you! Throughout this thesis I received financial support from the foundation for cultural heritage (PATRIMA) and I would like to thank them for the research grant and for the opportunity to obtain my PhD.

In this point I want to thank my advisors in my internship in LRI Nathalie Pernelle and Fatiha Saïs. They are the reason I came to Paris and also for starting this PhD.

This thesis would not be possible without my friends helping get through the difficult times. I would like to thank them for their emotional support, for all the happy moments we had together and for caring for me all this time. Thank you for everything Stam, Jim, George B, Fab, George M, Vincent, Tasos, Vasia, Alexandra, Sejla, Juan, Carmen, Damian, Jesus, Foivos.

Especially to my girls, Katerina, Nelly, Ioanna, Danai, Dimitra, Sofia, you are the friends one needs. Thank you for always being there, for your support, for the hours we spent discussing research and personal issues, for all the happy moments we had and for those that will come. Thank you for being my friends (especially after these four years of PhD ;))!

Further, I wish to thank a lot my uncle Stelios and Xenia who did this long trip for being present in my defense and support me in this special day of my life.

Words cannot always express the feelings for special persons that give meaning in your life, stands next to you in difficult personal moments and being happy for your happiness. Special thanks to Nik for believing in me, for his unconditional support, encouragement and his patience. He has been my source of strength, confidence and energy.

Finally, I dedicate this thesis to my family. My brother Nikos, always had the way to cheer me up and encourage me. My mom Maria, my very first friend, my person. My dad Orestis, my greatest supporter. They raised me, loved me, trusted me, supported me and being always there for me since the very first day of my life. To them, thank you for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Objectives

Web 2.0. The advent of the World Wide Web, especially Web 2.0, has led to drastic changes on how information is created and consumed. Information is accessible online and in a decentralized manner. To a large extent, the content is provided by Web users explicitly in platforms like Wikipedia, blogs, Social Media etc. The content creation is usually centered around entities, or subjects/topics that represent users' expertise.

Recent statistics show an increasing number of users on the Web [7], with rough estimates of more than 3 billion. Inherently, these users come from very diverse demographics, backgrounds, level of expertise etc. As a consequence the created content reflects such diversity, and with one of the most prevalent features representing a highly dynamic space [42, 9], with content constantly changing. Additionally, apart from content changes, the structure of such pages varies greatly [9] (e.g. HTML or XML structure which is used to represent the content).

The successful growth of Web 2.0 is mainly attributed to the interaction between users and Web applications, creating an online ecosystems, with users being active providers of emerging and new information online. Notable examples, include Wikipedia [1], as one of the largest online encyclopedias that is collaboratively created by Web users. It consists of representation of entities in 283 languages, with varying degree of completeness, with English Wikipedia being the largest collection. Other examples like IMBD [2] consisting of movies and user reviews about movies. As such they represent highly dynamic datasets with content and structural changes.

Furthermore, more advanced scenarios include Web services, which represent applications hosted on remote servers that support complex operations executed over datasets, hence, providing dynamic and on demand access for Web users or other Web applications. For example, well-known services like Amazon Web services [3], eBay [4], or MusicBrainz [5],

1. Wikipedia is in top–10 ranking of most visited pages based on ALEXA [6]
2. `http://www.imdb.com`
3. `https://aws.amazon.com`
4. `https://go.developer.ebay.com`
5. `http://musicbrainz.org`

represent large-scale datasets, in the case of Amazon and eBay, with product information coming from single users and large retailers, whereas in the case of MusicBrainz, with a constantly evolving dataset about music in general. The operations that provide dynamic and on-demand access in this case are necessary for multiple reasons, i.e., prices of products change, availability of products varies, and in the case of MusicBrainz, artists publish new records etc.

The wealth of the Web, respectively its applications and user base has gathered much attention from major Internet companies like Google, Microsoft or Yahoo!. For example, Google, has already incorporated a vast number of Web services (e.g. *flights, calendar* etc.). Furthermore, integrating many entity-centric sources like Wikipedia and IMDB, who provide information in the form of facts about an entity (e.g. birthdate of a movie producer), *related information* (e.g. all movies of a producer) [38]. As such these are noted as *linked data* or also referred as *structured data*.

However, as to now, apart from few use cases that have integrated only specific sources, there are no means on accessing such wealth of Web sources in a unified manner. The problem of integrating Web sources remains an open challenge. The challenges arise due to the scale and dynamics, in terms of changes in content and schema used to represent such sources. Other challenges are the limited access to such sources (e.g. Web services have limited calls per user, and linked datasets return only a limited number of triples for a query), and in many cases accessing the full content is expensive in terms of time, storage, and network bandwidth. In the following subsections, we describe the challenges associated with two types of sources, namely *linked data* and *Web services*.

1.1.1 Linked Data

Functionalities as the ones provided by Google or other major industry players, are preceded by research in fields like Semantic Web. The main vision of Semantic Web [17] is to represent the Web content as structured data, and to provide a reasoning framework on this data, where, the meaning of a specific piece of information on the Web is formally represented in RDF. RDF data is machine readable and interpretable by both humans and machines (e.g. Web applications).

A successful example of the Semantic Web is the Linked Data (LD) initiative, which states four rules on how to represent formally content on the Web [20]. Those rules are: (i) one should use URIs as names for things, (ii) URIs should be represented through HTTP so that people can look up those names, (iii) useful information should be provided on URI looks up by using the standards (RDF, SPARQL), (iv) include links to other related URIs so that one can discover more things. Since the advent of LD, there has been a major shift on how information is published on the Web. Individual data providers (i.e. governmental organizations, individuals) expose their data following the linked data principles. This has led to successful projects like Linked Open Data [6] (LOD), with thousands of datasets and billions of facts, from various domains like *government, health, education* etc.

However, some of the most prominent datasets exposed as linked data, are knowledge bases like DBpedia [13] and YAGO [103]. These represent knowledge bases that are

6. `http://lod-cloud.net/`

extracted automatically from Wikipedia using Information Extraction techniques. Furthermore, apart from Wikipedia alone, other Web sources like WordNet [5] (a dataset consisting of different senses of English words) are fused into such knowledge bases. In order to maintain the accuracy of extracted facts from Wikipedia, these approaches extract information only from a small subset of Wikipedia, namely its *infoboxes* and the *Wikipedia category structure*. Such limitations in terms of sources used to construct real-world knowledge bases, impact greatly the completeness of knowledge bases. Recent work in [96] propose the use of Web tables for knowledge base augmentation.

The multitude of automatically constructed knowledge bases has shown the importance of fields like entity resolution (ER) [27]. The aim of ER is to determine the multiple representation of an entity from different Web sources or Knowledge Bases. These aspects are especially important in the case of Linked Datasets, where the data is decentralized and no dataset has full coverage of facts regarding an entity. For instance, in Freebase 75% of entities of type Person do not have a *nationality* attribute, or 71% without a *place of birth* [22].

Therefore, approaches that align linked datasets at the schema level can be seen as complementary to the field of ER, and consequentially address the problem of *coverage, completeness,* and *relevance* of linked datasets in general [7]. However, as argued in [27], existing *ontology matching* approaches are not applicable at Web scale. Hence, important aspects like the efficiency of state-of-the-art approaches need to be addressed in order to be applied at Web scale. The advantages of such alignments apart from the *coverage* and *completeness* can be seen as precursors for instance matching [45].

Finally, apart from the automatically extracted knowledge bases, other approaches rely on collaborative efforts on constructing knowledge bases. For instance, Freebase[2] is one prominent example of a collaboratively created knowledge base. Nowadays, this has gathered much attention with initiatives like Wikidata [4], with more than 19 million (entities and other concepts) and a editor base of more than 2 million users.

Objective. Yet, despite the progress on publishing data following the LD principles, there are still challenges which hinder the fulfillment of the main vision of Semantic Web for an interconnected data landscape. The main unresolved challenge, yet to be addressed, deals with the heterogeneity of schemas used to represent content in such datasets. In this thesis, we address the problem of aligning linked datasets, namely *relation alignment* in disparate schemas in Chapter 4. Relation alignment for linked data, allows for *query rewriting* by rewriting *relation atoms* of a query, from a source dataset into the aligned (subsumed) relations of another dataset. Hence, allowing access to information about a specific relation across linked datasets through the computed alignments, and furthermore, without the need of class alignment.

7. The *nationality* attribute of an entity of type Person from Freebase can be complemented by the facts coming from DBpedia.

Challenges

Schema Heterogeneity. Figure 1.1 shows the LOD, where the connections between datasets consist of common entity instances. While alignment of linked datasets at instance level is largely addressed, alignment at schema level remains largely unresolved. In LOD there are roughly 650 schemas [125] used to represent the datasets. In many cases, there are multiple definitions for the same classes/concepts at different schemas. The differences are in two aspects: (i) *linguistic*, where data providers use different naming for the same concept, e.g. `Car` vs `Vehicle` or in the case of relations *gender* vs. *genderOf*, and (ii) *structural*, differences at representation level, e.g., a relation describing *gender*, in one case can have a literal value and in another a concept defining *gender*.

In a study [30], the authors analyze a specific category of linked datasets, namely that of *education*. An interesting observation is the heterogeneity of schemas used to represent educational information, in many cases, there were found multiple schemas defining similar concepts. In majority of cases these were not properly aligned. Hence, the problem of schema or ontology alignment has attracted attention from research in the field of ontology and schema matching [94, 99]. The proposed approaches to a large extent have made use of schema and instances to perform ontology alignment. Yet, relation alignment still remains an open challenge.

Dataset Accessibility. Exposing data as Linked Data, providers should give means of accessing the data. There are several approaches to expose RDF data in LOD, that are mostly subject to the preferences of data providers. Some of the most common access methods used at the moment in LOD are: (i) *RDF Dumps* where users can download and store locally entire datasets. Nevertheless, such a solution where data is stored locally does not scale up and raises the problem of freshness of the data, and (ii) *SPARQL endpoints* allowing for greater flexibility for data access through SPARQL queries.

The means to access linked datasets poses a significant challenge on carrying out tasks like schema matching, specifically in our case *relation alignment*. Due to the fact that there has been an exponential increase on the number of datasets exposed as linked data, such tasks cannot be applied on dataset snapshots for two reasons: (i) linked datasets in LOD nowadays count more than 30 billion triples [3], and (ii) datasets constantly evolve, hence, alignments created at one snapshot might not hold on a different version of a dataset.

Therefore, in Chapter 4 we address such efficiency issues and perform the task of relation alignment in an online setting. We query the datasets through their SPARQL endpoints and extract *small samples* of the data with the main objective of maximizing the coverage of discovered relation alignments, while performing the task with high accuracy and efficiency.

8. `http://lod-cloud.net/versions/2014-08-30/lod-cloud_colored.png`

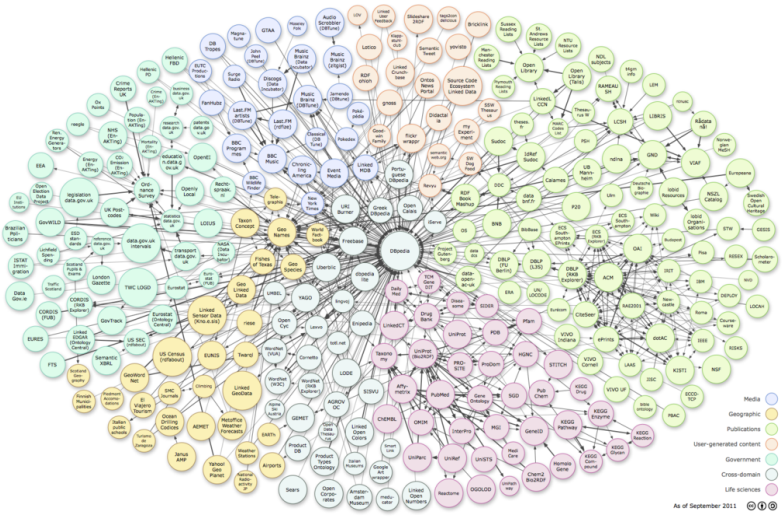


Figure 1.1: Linked Open Data Cloud in 2014. [8].

1.1.2 Web Services

While through linked datasets applications and users are able to inter-link information about entities and specific topics across different datasets, a large number of data providers choose to make their data accessible through Web services. This presents an additional valuable set of resources which are up to date and are accessible on demand.

In recent years, several important content providers such as *Amazon*, *MusicBrainz*, *IMDB*, *GeoNames*, *Google*, and *Twitter* have chosen to export their data through Web services. These services cover a large variety of domains: books, music, movies, geographic databases, transportation networks, social media, even personal data. This trend gained momentum thanks to the Open Data Initiative, the success of mash-up applications, and new initiatives that grant users programmatic access to their personal data. Users can access the data by invoking services, but they can hardly copy the entire content of the remote source. Hence, a Web service seems to be a sweet-spot in the trade-off between data sharing and protection. Currently there are more than 15,000 [41] content provider who choose Web services as a way to share the data.

The wealth of data exported through Web services presents opportunities for the development of new intelligent applications which seamlessly integrate multiple services. Consider a personalized application that proposes vacation plans. A detailed example is explained below.

Example 1. *Consider a personalized application, a trip planner (depicted in Figure 1.2). A user wants to use this application to organize a trip to a musical event. The applications combines several Web services. First of all, based on user's music preferences, which is extracted from from a social Web service like* Facebook *API or* Twitter *API; it further retrieves offers for concert tickets from a ticketing Web service like* EventIm. *Ideally, based on the retrieved musical artist or band profile from the Web service* MusicBrainz, *it checks the newest records or albums, and further filters the concert tickets accordingly. Dependent on the location of the user, it calculates the best route to the closest concert*

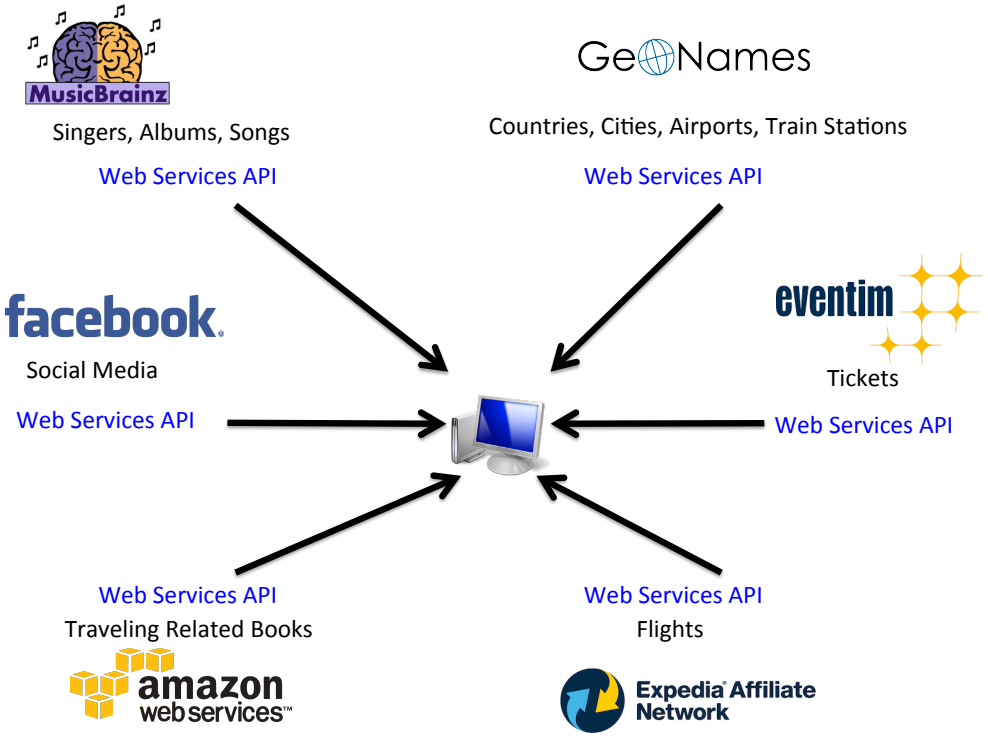


Figure 1.2: Application example: automatic computation of vacation plans.

venue from GeoNames.

Objective. In this thesis in Chapter 3, we envision the integration of Web services, such that the output from any possible operation call from an arbitrary Web service can be interpreted uniformly through a global schema from real-world knowledge bases like DBpedia or YAGO. For each operation call, we construct a *view with binding patterns* over the global schema. The immediate advantages allow applications as the one envisioned in Example 1 to exploit the wealth of existing Web services.

Challenges

However, considering the wealth of Web services, a fundamental issue, namely the integration of Web services still remains. Despite the research progress in Web service representation [85, 10, 58, 58, 101] and discovery [40, 100, 24]; integration is mainly hindered by the lack of explicit schemas for the output of Web service operations. Failing to fully integrate Web services at the output level leaves the landscape largely disintegrated, with the applications being developed only for specific services.

The challenging factors in integration output from REST Web services is that they do not provide information about the schema use to represent the output. Furthermore, we focus on REST Web services as they represent one of the most common Web services in contrast to its competing standard SOAP [41].

Other challenges we encounter from the large number of services, is the heterogeneity of output representation, and in many cases, the limited number of Web service calls.

1.2 Contributions

The contributions of this thesis is a framework that provides a unified model for accessing the wealth of linked data and Web services. It is part of a project called *EDOP* of the *foundation for cultural heritage* (PATRIMA [9]). The aim of this project is to design and create a cultural heritage dataspace from RDF knowledge bases or Web services.

As part of our contributions, we address two issues related to Web services and linked data: (i) mapping of Web Service output into a global schema, and (ii) online relation alignment of RDF datasets. Both are combined under the scope of EDOP project. Figure 1.3 shows an overview of the architecture of this project composed of the individual components from the systems developed as part of our contributions.

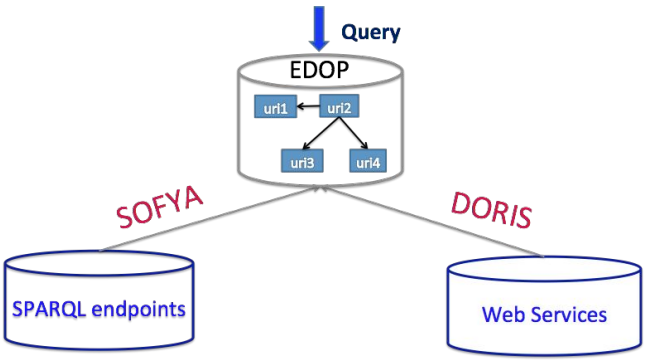


Figure 1.3: EDOP project basic architecture.

DORIS: Discovering Ontological Relations In Web Services. The first contribution of this thesis is our approach, called *DORIS*, on mapping the output of Web services into a global schema. It enables uniform access to the data encoded by Web services using the REST architecture. In this way we are able to integrate different Web services, respectively their output.

In DORIS, to map the output of Web service calls, we adopt an an *instance-based* schema mapping approach due to the fact that Web services using the REST architecture do not provide schema information. We follow the intuition where exploit the *intersection* of Web service call results and *literals* in a knowledge base. In this way, we automatically model a Web service as a *view with binding patterns* over a global RDF schema (extracted from a knowledge base). Finally, we provide a transformation function in order to transform the output from a Web service call into the specific global schema.

9. `http://www.sciences-patrimoine.org/index.php/homepage.html`

SOFYA: Online Relation Alignment for Linked Datasets. The second contribution is an online instance-based relation alignment approach for RDF knowledge bases, called *SOFYA*. Unlike most of the ontology alignment approaches we focus on relation alignment (instead of class alignment) which can be used directly on *query rewriting* for data integration. For a given relation from a knowledge base, we uncover relations from a target knowledge base that can be aligned. We consider a specific case of alignment, namely that of relation subsumption, hence, a relation from a target knowledge base should subsume our starting relation.

We propose a supervised machine learning relation alignment model, which we setup in an online setting. Specifically, we do not need local access on the data, all the computations are made on data that are queried from the knowledge bases.

For our approach to be efficient in an online setting, we sample for entity instances with the objective of improving the efficiency in terms of query-execution overhead (time and network bandwidth usage) and coverage of candidate relations.

We carry out an extensive experimental evaluation on three real-world knowledge bases: DBpedia, YAGO, and Freebase, we assess our approach for its *effectiveness* and *efficiency* of uncovering relation alignments.

1.3 Thesis Outline

In the following, we provide an overview on the structure of the thesis, and outline the main contributions in each Chapter.

Chapter 2 This chapter presents the necessary background information that is required in the context of this thesis. Furthermore we define notions that are used throughout this thesis. In the second part of this chapter we provide an analysis of the state-of-the-art approaches in the area of schema matching as well as in related to data extraction from Web services areas.

Chapter 3 This chapter describes the first contribution of this thesis, our approach on mapping the output of Web services into a global schema, called DORIS. This chapter is based on the articles published at top-tier conference CIKM [10] [60] and the respective national conference publication [63]. The DORIS system was demonstrated in a national [61] and at the international conference ISWC [11] [62]. The contributions in this chapter are:

- An algorithm that provides a formal description of the output of a given Web service in terms of a given global schema.
- A transformation function, as a script, that will transform the output of a Web service in terms of a global schema.

10. http://www.cikm-2015.org
11. http://iswc2015.semanticweb.org

- An algorithm that discovers Input/Output dependencies between Web services of the same API.
- An extensive experimental evaluation on real Web services coming from four different domains and on three real RDF knowledge bases (DBpedia, Yago and BNF), validating the performance of our approach.

Chapter 4 This chapter describes the second contribution of this thesis. Part of this chapter follows the international conference publication EDBT [12] [59]. More specifically the contributions in this chapter are:

- An instance-base relation alignment algorithm that discovers subsumpsion relationships between knowledge base relations.
- A supervised machine learning model that combines a set of light-weight features to decide if the subsumption relationship is *correct* or *incorrect*.
- An efficient version of the algorithm that uses a small sample of the data (samples) to produce the subsumptions.
- An extensive experimental evaluation of our method on three well-know knowledge bases DBpedia, YAGO, and Freebase.

Chapter 5 Concludes and provides a discussion about possible future directions as well as some ongoing work.

12. `http://edbticdt2016.labri.fr`

Chapter 2

Background and State-of-the-art

In this Chapter, we describe the background information needed to present the work in this thesis, and further review state-of-the-art approaches that are related with the contribution in this thesis.

The Chapter is organized as following. Section 2.1 discusses the Resource Description Framework (RDF) [120] data model which is used to represent knowledge bases. Furthermore, it is one of the core concepts in Semantic Web. In addition, we describe SPARQL query language functionalities [117], which we use as means to access and query RDF data. Section 2.2 introduces an abstraction of a Web service and Web service APIs, and explain their usage in this work. Additionally, we define a language that is used to query and transform the results from a Web service. Section 2.3 presents the integration model we use in Chapter 3, where we present our approach on mapping the output of Web services into a global schema.

Finally, in Section 2.4 we present a detailed review of related literature on the two main research contributions in this thesis with respect to the approaches in Chapter 3 and Chapter 4.

2.1 Knowledge Bases

We see a knowledge base (KB) as a structured dataset represented in RDF. In this section we present an abstraction of the RDF data model, that is used to represent knowledge bases. Furthermore, we describe the features of SPARQL 1.1 query language, which we use to query KBs or RDF datasets through their respective SPARQL endpoints. As it will become evident in the next sections (see Chapter 4), we will use SPARQL endpoints to perform the task of ontology relation alignment for any given KB pair.

2.1.1 RDF Data

The **R**esource **D**escription **F**ramework (RDF) [120] is a graph data model proposed by W3C as the standard for knowledge representation on the Semantic Web. In the RDF data model, we assume a set of *entities*, a set of *literals*, a set of binary *relations* and a set of *classes*, that are described as follows:

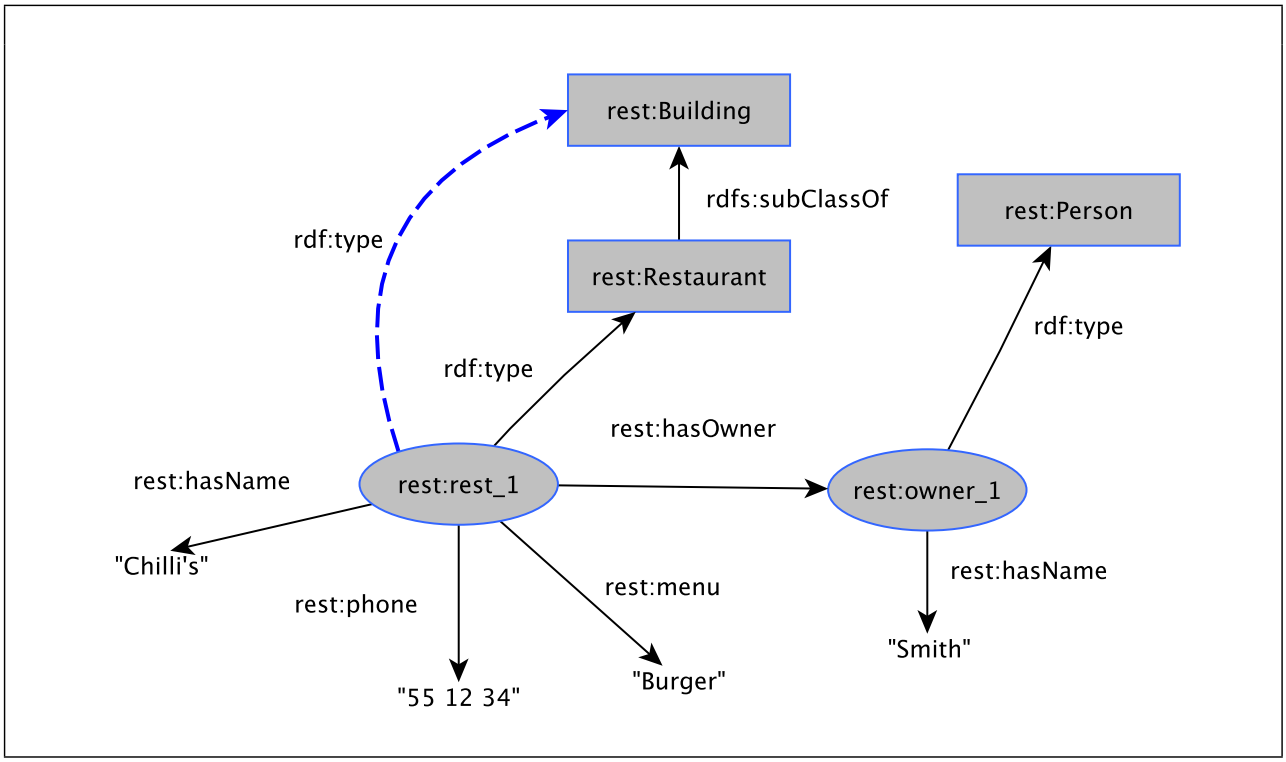
- **Resources (R):** A resource is a real-world entity such as a person, an organization, a book, or an abstract concept. In RDF, a resource is uniquely identified by a Uniform Resource Identifier (URI). We will refer to a resource also using the term *entity*.
- **Literals (L):** A literal is a value like a string, a date, or a number.
- **Relations (P):** A relation (or property) holds between two entities or between an entity and a literal. An element of a relation is called *fact*, and we write $r(x, y)$ to say that the entity with URI x stands in the relation r with the entity or the literal y. RDF data is organised in triples of form $\langle$*subject, predicate, object*$\rangle$ where the *subject* x is in a relation r to *object* y. The *domain* of a relation is the class from which all the first arguments of its facts are taken. Analogously, the *range* is the class of the second argument.
- **Classes (C):** A class corresponds to a set of entities, such as the class of `Singers` or the class of `Cities`. This class membership is expressed as facts of the relation `rdf:type`, which is part of the RDF specification. The classes can be organised in a hierarchy using the RDFS [121] relation `rdfs:subClassOf`. By inference, the set of instances of a class includes the set of instances of its sub-classes, defined in Equation 2.1

$$\begin{aligned} &\forall x, c_1, c_2 : \\ &\qquad \texttt{rdfs:subClassOf}(c_1, c_2) \wedge \texttt{rdf:type}(x, c_1) \\ &\qquad \Rightarrow \texttt{rdf:type}(x, c_2) \end{aligned} \tag{2.1}$$

Definition 2.1.1 (Knowledge Base (KB))**.** *Given a set of resources R, a set of relations P and a set of literals L, a knowledge base $KB(R, P, L)$ is a set of triples $\langle x\ r\ y\rangle$ from $R \times P \times (R \cup L)$*

Whenever R, P and L are understood from the context, we will simply write KB instead of $KB(R, P, L)$.

Graph Representation. Conceptually, an RDF knowledge base can be represented as a directed graph, where the arguments of the facts map to nodes and the facts to directed edges. More precisely, each fact $r(x, y)$ maps to a directed edge where x is the source node, y is the target node, and r the relation name is the label of the edge.




```
rest:hasName(rest:rest_1, "Chilli's")    rest:hasOwner(rest:rest_1, rest:owner_1)
rest:phone(rest:rest_1, "55 12 34")      rdf:type(rest:owner_1, rest:Person)
rest:menu(rest:rest_1, "Burger")         rest:hasName(rest:owner_1, "Smith")
rdf:type(rest:rest_1, rest:Restaurant)
rdfs:subClassOf(rest:Restaurant, rest:Building)
```

Figure 2.1: Knowledge base example: graph representation (upper part) and the triple set representation (lower part).

Example 2. *Figure 2.1 shows an example RDF knowledge base containing data about restaurants in New York City. The upper part shows the graph abstraction. The circled nodes represent resources (or entities), while the rectangles represent classes. The literal nodes are encoded in quotation marks. The labeled edges represent the relations holding between the entities or the entities and the literals of the knowledge base.*

In the bottom part of the figure, the knowledge base is shown as a set of triples. Note that the dotted blue arrow (standing for the triple `rdf:type`*(rest:rest_1, rest:Building), does not appear in this set. Yet, the fact can be inferred from the inference rule (Equation 1.1) and the facts* `rdf:type`*(rest:rest_1, rest:Restaurant) and* `rdfs:subClassOf` *(rest:Restaurant, rest:Building).*

Entity-Entity vs. Entity-Literal Relations. We can classify the relations depending on the type of the objects in their statements. While we can have only entities (R) as subjects, the objects can be either entities or literals ($R \cup L$). In this work, we assume that for any relation, the range is either a set of entities or a set of literals and not a combination of

both entities and literals. Based on this, we consider the following classification:

Definition 2.1.2 (*EE and EL* Relations)**.** *Given a knowledge base $KB(R, P, L)$ and a relation $r \in R$, we say that r is an entity-entity relation, or simply r_{ee}, if $\forall r(x, y) \in KB \Rightarrow y \in R$. r is an entity-literal relation (r_{el}) if $\forall r(x, y) \in KB \Rightarrow y \in L$.*

Additionally, we define for every *entity-entity* relation r, the *inverse* relation r^- , as follows.

Definition 2.1.3 (Inverse Relation)**.** *A relation is called the* inverse *of a relation r, written r^-, if $\forall x, y : r(x, y) \Leftrightarrow r^-(y, x)$.*

In this work, we assume that any knowledge base stores for every *entity-entity* relation r, all the facts of the inverse relation r^- that are obtained from the facts of r by swapping the subject and the object. The new facts conform to the RDF constraint of having entities on subject positions.

Relation Functionality. A relation is *functional* if there are no two facts of the relation that share the same subject and have different object values. However, real-world KBs are noisy, and may erroneously store more than one distinct facts of a functional relation for one subject. Therefore, in line with [102], we define the functionality, as follows:

$$func(r) := \frac{\#x : \exists y : r(x, y)}{\#(x, y) : r(x, y)} \tag{2.2}$$

where the value in the nominator, $\#x$, is the number of values for x for relation r, whereas the denominator represents the number of statements or facts for all the subjects x. A perfect functional relation will have a functionality of 1 (if the knowledge base does not contain wrong facts).

Equivalent Resources. In RDF, specifically, the definition of URI from the sets R, P are meant to be global. This means that the same URI can be used by different KBs to define identical resources or relations. Conversely, two different resources can refer to the same real world object. For example, US and USA can both refer to the United States of America. Such resources are called *equivalent*. In structured data, the equivalence between two resources or entities is expressed through `owl:sameAs` statements. Hence, we can write `owl:sameAs`(US, USA). For simplicity, in this work, we write $US \equiv USA$.

The two entities do not necessarily belong to the same knowledge base. Usually, different URIs representing the same real world entity occur mostly in cases where the entity is part of different knowledge bases that have been constructed independently from one another. In this way `owl:sameAs` represents the statements that connect different knowledge bases that are part of the Linked Open Data [1] project. According to [97], `owl:sameAs` is the most frequently used linking predicate in the majority of the topical domains in the Linked Open Data.

1. `http://lod-cloud.net/`

2.1.2 Query language

The standard for querying RDF datasets is SPARQL [117]. It was proposed in 2008, and since then it is a W3C standard. Since then SPARQL has evolved (SPARQL 1.1 [118]) by supporting new features that brings it closer to complex relational query languages like SQL. SPARQL can be used to query both data and schema.

A SPARQL expression can take one of the following forms:

- **SELECT:** Returns row data bindings to the variables which participate in the query. A variable in SPARQL is not bound to any predefined value but is subject to the result of the query (e.g. *?birthDate*)
- **CONSTRUCT:** Returns an RDF graph constructed by the the result triples that match the query.
- **DESCRIBE:** Return an RDF graph that describes the resources that match.
- **ASK:** Returns a boolean value whether the query pattern matches or not.

In this work we consider only **SELECT** SPARQL queries.

SPARQL is based on the concept of matching graph patterns. In this thesis we consider a subset of SPARQL named *basic graph pattern* (BGP) queries. BGP consists of a set of simple graph patterns, specifically the *triple patterns*. A triple pattern is similar to an RDF triple, with the distinction that *variables* are allowed on each of the three positions, that is, the subject, predicate or object positions. A *variable* in SPARQL has as a prefix the symbol '?'. More formally we define a triple pattern as:

Definition 2.1.4 (Triple Pattern)**.** *Let U be a set of URIs, L be a set of literals and V be a set of variables, a* triple pattern *is a tuple $(s\ p\ o)$ from $(U \cup V) \times (U \cup V) \times (U \cup L \cup V)$.*

Using the previous definition of a triple patter we define the BGP query as following:

Definition 2.1.5 (BGP Query)**.** *A* BGP query *is a conjunctive query of the form*

$$\texttt{SELECT ?x}_1\texttt{...?x}_m\ \texttt{WHERE \{ t}_1 \wedge \texttt{t}_2 \wedge \ldots \wedge \texttt{t}_n\ \texttt{\}}$$

where $\texttt{t}_1 \ldots \texttt{t}_n$ are triple patterns and $\texttt{?x}_1 \ldots \texttt{?x}_m$ are distinguished variables appearing in $\texttt{t}_1 \ldots \texttt{t}_n$.

The `SELECT` clause specifies the variables whose bindings should appear in the result. SRPARQL engines can be used to query at once several graphs (KBs) stored in an RDF store. The `FROM` clause is used to specify the graph that will be used to answer the query. If the `FROM` clause does not exist, then the query will be executed over all triples in a given RDF store. The `WHERE` clause contains all the triple patterns, which represent the conjunction of conditions that must be met in order to obtain the desired result. We will refer to the set of the triples patterns as using the term graph pattern.

Let an embedding of the graph pattern be a subgraph of the KB such that every variable in the query is bound to a resource, predicate or literal. For every embedding of the graph pattern, a tuple consisting of the bindings of the variables in the `SELECT` clause is returned as solution. Hence, the query result is an ordered list of tuple bindings.

```
PREFIX rest:  <http://restaurants_new_york.com/>
SELECT ?x ?y
WHERE {
      ?x rdf:type rest:Restaurant
      ?x rest:hasOwner ?y
}
```

Figure 2.2: Example SPARQL query Q_1.

?x	?y
rest:rest_1	rest:owner_1

Figure 2.3: Result of SPARQL query Q_1.

Example 3. *For example, consider the query depicted in Figure 2.2. The execution of the query over the knowledge base of Figure 2.1, leads to the result shown in Figure 2.3. The result set is illustrated in a tabular form. Each solution is shown as one row in the table.*

To enforce a specific ordering or sorting of the result set, SPARQL provides several modifiers. These modifiers range from simple sorting, distinct solutions in the result set, and to finally specifying the desired amount of answers. The *Solution Sequence Modifiers* used in this thesis are the following:

- **DISTINCT:** Ensures that the returned solutions in the sequence are unique.
- **ORDER BY:** Puts the solutions in specific order.
- **LIMIT:** Restricts the number of the returned solutions.
- **OFFSET:** Causes that the returned solutions will start after a specific number of solutions. When the `offset` is zero the returned solutions will start from the first solution.

VALUES Construct. An interesting feature provided by SPARQL 1.1 which we use in Chapter 4 is the `VALUES` construct. Using `VALUES`, data can be directly written in a graph pattern and added to a query. In this way we can `JOIN` in-line data with the results of a BGP query executed over a given KB. This is a useful construct when we the results of two BGP queries need to be joined and are executed by two remote SPARQL engines. In order to reduce the data transfers, the results of one query can be included in the second query. An example query using the `VALUES` feature is depicted in Figure 2.4. In this example the result solutions for the variable `?x` will be `joined` with the values of `?y` provided in the `VALUES` clause.

FILTER Constraint. SPARQL also gives the capability for adding constraints to the variables included in the graph pattern using the `FILTER` clause. In this way, answers are

```
PREFIX rest:  <http://restaurants_new_york.com/>
SELECT ?x ?y
WHERE {
VALUES (?y) {
      rest:owner_1
      rest:owner_2
}
{ SELECT ?x ?y WHERE {
      ?x rdf:type rest:Restaurant
      ?x rest:hasOwner ?y }
}
}
```

Figure 2.4: Example SPARQL query with `VALUES` Q_2.

restricted to those which result in evaluating the filter expression to true. The constraints that can be included in the `FILTER` clause may assign a specific value to a resource or to introduce arithmetic constraints to literals or string similarity functions.

In the rest of the thesis we will use the terms SPARQL query or simply query, referring to the SPARQL fragment described by Definition 2.1.5 along with all the described constraints / features.

2.1.3 Knowledge Base Paths

In this work, we are interested in the sequence of relation names that labels a path linking two entities or an entity to a literal. While the source entity is always given, the target entity (or literal) might be required to be returned. If the two entities are given, then we talk about boolean KB paths, defined as follows:

Definition 2.1.6 (Boolean KB Path)**.** *Given a knowledge base $KB(R, P, L)$, and the relations $r_1, \ldots, r_n \in P$, we write $r_1.....r_n(x, y)$ to say that there is a* path *in KB with relation names $r_1, ..., r_n$ between node x and node y that does not visit any node twice:*

$$\exists x_1, ..., x_{n-1} : r_1(x, x_1) \wedge ... \wedge r_n(x_{n-1}, y) \wedge |\{x_1, ..., x_{n-1}\}| = n - 1$$

Definition 2.1.7 (KB Path)**.** *We write $r_1.....r_n(x) := \{y : r_1.....r_n(x, y)\}$ to indicate all entities or literals reachable by the path from x.*

For simplicity, we will use p for a *Boolean KB path* or a *KB path* of one or several relation names.

This gives us a simple language, compatible with SPARQL, that can express paths both in the Web service call result (Section 2.2.4) and in the knowledge base.

2.2 Web Services

According to W3C, a *Web service* (WS) [109] is a software system designed to support interoperable machine-to-machine interaction over a network. This interaction usually concerns the exchanging of data between applications or systems.

In 2002, the W3C Web service Architecture working group defined a Web service architecture where a Web service interacts with other systems using SOAP messages. and the input as well as the output of the Web service is formally described in an XML-based machine-processable format namely WSDL (Web Services Description Language).

In 2004, the W3C [111] proposed a new category of Web services, the *REST-compliant* Web services. The purpose of REST Web services is to manage Web resources representations by using a set of operations. REST uses standard HTTP which is much simpler than SOAP thus, REST has better performance and scalability. Major e-commerce and Internet-related companies like Amazon, eBay, Yahoo and Google are in favor of REST Web services. Moreover, 80% of the Web services [41] use the REST architecture.

In this work we consider only REST Web services, from now on whenever we refer to *Web services* or *WS*.

2.2.1 REST Architecture

A REST Web service follows a standard client-server architecture. Figure 2.5 illustrates a request-response communication between a *Client* and a *Service Provider* where the client sends a *call* request to the service provider. The call takes the form of the parameterized URL that includes the values of the input parameters. More precisely, the URL is a case insensitive string of form:

scheme://host:port/path?queryString#fragment

Where:

- *scheme* is the protocol, usually HTTP, but it can also be FTP or HTTPS;
- *host* is either a DNS name or a IP address;
- *port* is the port. If missing, the default port is 80 when the scheme is HTTP;
- *path* is a set of text segments delimited by the "/";
- *queryString* is a list of parameters represented as *name=value* pairs, separated by the character *&*;
- *fragment* is an identifier used to point to a particular place in the document.

Example 4. *Consider the URL depicted in Figure 2.5. The scheme, the host and the path are "HTTP", "www.musicbrainz.org", and "ws/2/artist/" respectively. The port is not given, hence it is considered to be the default port 80. The URL consists of only one parameter: "query=Madonna".*

We define a Web service call as following:

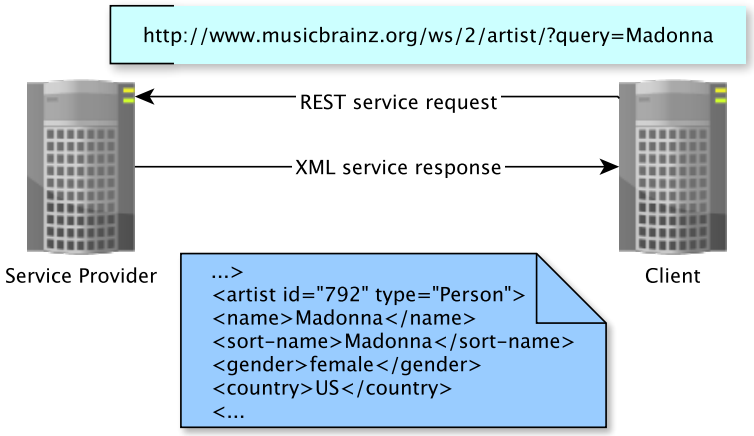


Figure 2.5: REST Web service architecture.

Definition 2.2.1 (WS call)**.** *Given a Web service f a Web service call is a parameterized URL that is sent from the Client to the Service Provider through HTTP.*

Following the request (WS call) from the Client, the Service Provider computes a *call result* for the given call and sends back to the client an encoding of it according to (1) a language (typically, XML or JSON) and (2) a schema (typically, unknown). As encoding language, we consider only XML. This is without loss of generality as we will show next JSON fragments can be translated into XML documents. In this work, we assume that the Web service encodes all its results using the same schema.

Furthermore, we deal only with Web services that provide means on accessing the underlying data sources, e.g. MusicBrainz. These services can be seen as the execution of a predefined parameterized query over the remote source. Note that, with a Web service one cannot execute any query on the remote source. Only parametrized and predefined queries for a Web service can be executed. In order to address this limitation, Web sources typically publish not one but several Web services that form what is called a *Web service API*.

2.2.2 Call Results

The Web services implementing the REST architecture are not constrained to follow a specify language for encoding their results. However, most of them are using XML [116] or JSON [119]. The two languages are similar as the underlying model for both of them is an abstract tree.

XML. E**X**tensible **M**arkup **L**anguage (XML) is a markup language for encoding Web documents in human and machine readable format. XML is a W3C standard [116] and is considered as the de facto standard for data exchange across the Web. The wide adoption of XML is due to its simple, flexible and generic data model. Moreover, since the data is easily interpretable by both humans and machines due to its semi-structured nature, as

```
<root>
 <Restaurant>
  <Name>Chili's</Name>
  <Entree>Burger</Entree>
  <Phone>555-1234</Phone>
  <Owner>Smith</Owner>
 </Restaurant>
 <Restaurant>
  <Manager>Smith</Manager>
  <Name>Darbar</Name>
  <Entree>Lamb Curry</Entree>
  <Entree price="23">Beef Curry
  </Entree>
 </Restaurant>
 <Bar>Rose & Crown</Bar>
</root>
```

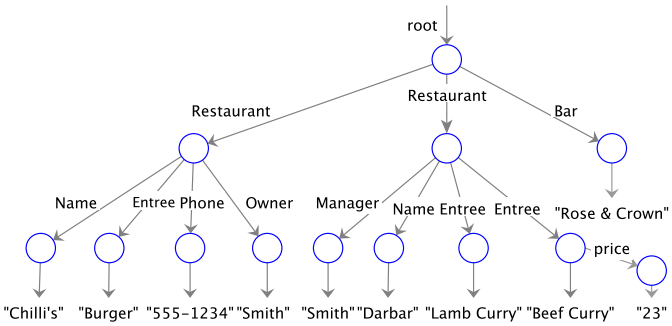


Figure 2.6: Example XML document: XML encoding (left) and abstract tree representation of the same document (right).

such it makes it a natural choice for representing and integrating data that do not obey strict schema definitions.

In this work we are not interested in the full functionality that XML language provides. Subsequently we present a subset of XML functionalities that are used in this thesis.

XML is organized in documents and the encoded data forms a tree structure. In line with the W3C standards, in this thesis, we represent an XML document as a rooted, labeled tree. In XML documents, each node can be an *element*, a *text node*, or an *attribute*. An XML *element* node may contain a list of (sub-)elements, text nodes and/or attributes as children. A *textual node* contains text encoded in *Unicode* format and can be of any arbitrary length, and an *attribute* node has a *label* and a *value* assigned for it.

We make the distinction between two kind of nodes. We call the *elements* or the *attributes* of an XML document *structure nodes*, and the attribute values or the contents of elements *text nodes*.

For every well-structured XML document there is exactly one *root* or a parent node that contains all other nodes or child nodes. Since the data forms a tree structure, each child node in an XML document has exactly one parent node, consequently a parent node can have multiple child nodes.

Example 5. *Figure 2.6 presents an example of an XML document (left) encoding metadata about* `Restaurants` *and* `Bars` *in New York. The corresponding XML tree is shown in the right part of the figure, where the structure nodes are depicted as circles while text nodes are shown in quotation marks.*

JSON. JavaScript Object Notation (JSON) [119] is a lightweight data-interchange text format. Easy for both humans and machines to read and generate. The basic structures used in JSON to encode data are *object*, *array*, and *value*. An object is an unordered set of name-value pairs beginning with left brace ({) and ending with a right brace (}). Each name is followed by colon (:) and the name-value pairs are separated by comma (,). An array is an ordered collection of values where a value can be a string, a number, an object, an array or one of the values: true, false or null.

The basic transformation rules that allow JSON data structures to be translated to XML documents are the following:

- JSON names become XML element names
- JSON object members and arrays become XML elements, and
- Simple values become XML text nodes

Example 6. *Consider the following JSON document:*
{
"city": "Paris",
"country": "France"
}

After the translation to XML will have the following result:

<root type="object">
<city>Paris</city>
<country>France</country>
</root>

For uniformity, in this work we assume that call results in JSON are transformed to XML documents. This can be done by standard tools that consider the aforementioned transformation rules.

2.2.3 DataGuides

One of the reasons of the widespread of XML, is that it does not enforce strict schemas over XML documents. Often, XML is referred to as a semi-structured data model, due to the absence of a fixed schema. Hence, in many cases XML documents might miss a formal definition of a schema. This main characteristic of XML along with the lack of a formal description for REST Web services prompted us to use *DataGuides*.

To account for the missing schemas and the advantages of having a formally defined schema over an XML document, Goldman and Widom [48] introduce the notion of *DataGuides*. A DataGuide is a concise, accurate and convenient structural summary of an XML document. More precisely, based on [48], in this work, we define a DataGuide as following:

Definition 2.2.2 (DataGuide)**.** *A DataGuide for a source XML tree s is an XML tree d such that every label path of s has exactly one label path instance in d, and every label path of d is a label path of s.*

Where a *labeled path* is defined as:

Definition 2.2.3 (labeled path)**.** *A labeled path of an XML node o is a sequence of one or more dot-separated labels, $l_1.l_2.\ldots.l_n$ such that we can traverse a path of n edges $(e_1 \ldots e_n)$ from o where edge e_i has label l_i.*

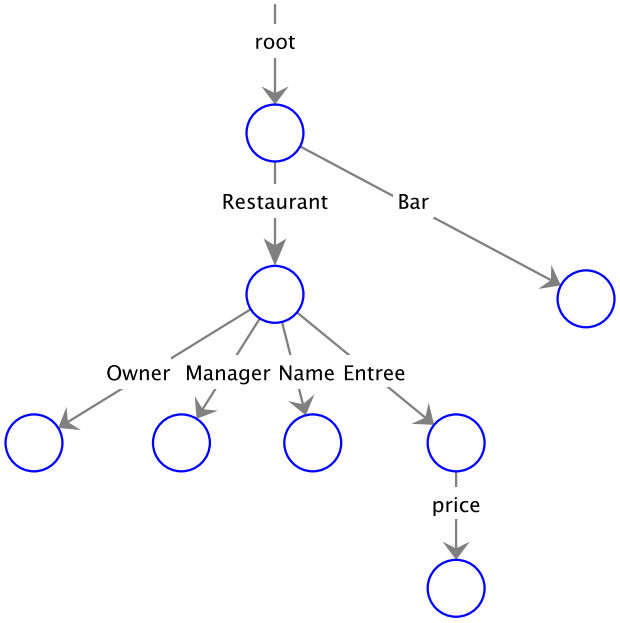


Figure 2.7: *DataGuide* for the *XML* document in Figure 2.6

Example 7. *Figure 2.7 shows an example of a DataGuide for the XML document presented in Figure 2.6. A label path of the XML tree starting from* `root` *node in Figure 2.6 is* `root.Restaurant.Name`.

Through *XPath* [114] expressions over a DataGuide we can check whether a path exists in the XML document. Hence, we can assert whether certain *structural nodes* exist in the document, or a specific ordering as parent–child nodes is possible in an XML document.

2.2.4 Querying Call Results

Given a Web service f, we refer to the root of the call result for x by $\lambda_f(x)$. We will omit the subscript f if it is clear from the context. We label an edge in the call result with the label of the target node of the edge. In Figure 2.6, e.g., the topmost edge will be labeled with "root". If an edge leads to a text node, we label the edge with the generic label τ.

Definition 2.2.4 (Boolean WS Path)**.** *We write $p(x, y)$ to say that there is a* path $l_1/.../l_n(x, y)$ *in the call result, with edge labels $l_1, ..., l_n$ between node x and node y.*

Definition 2.2.5 (WS Path)**.** *We write $p(x) := \{y : l_1/.../l_n(x, y)\}$ to mean all nodes reachable by that path from x.*

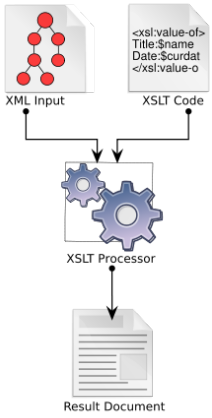


Figure 2.8: Process flow of Extensible Stylesheet Language Transformations (XSLT) [2].

This gives us a simple language that can express paths in the call result. This language is compatible with XPath (the language for querying XML documents), and with XSLT [115] (the language used for our transformation functions). To distinguish paths in the call result from paths in the KB, we use "/" for XML and "." for RDF data.

2.2.5 Transforming Call Results

The *WS paths* defined above, are used to create the *transformation function* which is provided by our system. More precisely, the transformation functions are expressed in Extensible Stylesheet Language Transformations (XSLT) [115], a declarative language based on XPath expressions. XSLT is used to transform XML-like documents to other XML-like documents. The original document remains unchanged and a new document is created using the content of the original one. The transformation process is presented in Figure 2.8.

2.3 Data Integration Model

Views With Binding Patterns. In one of the best known integration models, in LAV (local-as-view), a local schema is defined as a view (query) over the global schema. In our approach, on mapping the output of Web services into a global schema (DORIS – Chapter 3), the global schema is a given knowledge base and the local is a Web service. For this specific case of source with restricted access, we use the formalism of *views with binding patterns* described in [50].

Formally, a view with binding patterns is a conjunctive query of the form

$$q^{\bar{a}}(\bar{x}) \leftarrow r_1(\bar{x}_1), r_2(\bar{x}_2), \ldots, r_n(\bar{x}_n)$$

where $r_1, \ldots, r_n$ are relation names from the global KB and q is a new relation name. The tuples $\bar{x}_1, \bar{x}_2, \ldots \bar{x}_n$ contain either variables or constants. The query must be *safe*,

2. https://en.wikipedia.org/wiki/XSLT

i.e. $\bar{x} \subseteq \bar{x}_1, \bar{x}_2, \ldots \bar{x}_n$ (every variable in the head must appear also in the body). Every variable is adorned in $\bar{a}$ with a letter. We use i for input-output variables, and o for output variables. By input-output variable we mean that the result might return new bindings for that variable.

Example 8. *Consider a Web service encoding information related to restaurants and as global schema the knowledge base of the Figure 2.1. The Web service can be written as the following conjunctive query over a global schema:*
$getRestaurantInfo^{ioo}(x_1, x_2, x_3) \leftarrow hasOwner(x_1, x_2), menu(x_1, x_3)$

The aforementioned query defines how the Web service provider computes the function *getRestaurantInfo*. From the caller's point of view, the adornment *ioo* states that *getRestaurantInfo* can be called only if a value for x_1 is provided. The evaluation of the function call binds the variables x_2 and x_3 to the values that satisfy the conditions hasOwner(x_1, x_2) and menu(x_1, x_3).

2.4 State-of-the-art

In this section, we review related literature on state-of-the-art, relevant to the two problems we address in this thesis:

– *DORIS*. Mapping of Web service output into a global RDF schema

– *SOFYA*. Online alignment of ontological relations

In Section 2.4.1 we provide an overview of schema matching approaches. Schema matching is used in both parts of our contributions, namely in Chapter 3 and Chapter 4. In Chapter 3, we propose a schema matching approach to map the output from Web service operation calls into a global RDF schema, whereas in Chapter 4 we propose an online approach for ontological relation alignment.

Section 2.4.2 presents a detailed overview of schema matching approaches that are related to *SOFYA* approach in Chapter 4. Finally, in Section 2.4.3 we present related work for the approach in *DORIS* in Chapter 3. The related work is grouped into the following categories: (a) schema matching in Web services, (b) Query Discovery, (c) Wrapper induction, (d) Web service representation, and (e) Web service discovery.

2.4.1 Schema and Ontology Matching Approaches

In the following subsections we present an overview of the notions of *schema matching* and *ontology matching*. We introduce some of the main commonalities found across schema and ontology matching research, and their respective categorization (e.g. schema vs. instance based approaches.). These represent one of the largest corpora of related work that are closely related to the approaches in *DORIS* and *SOFYA*, Chapter 3 and Chapter 4, respectively.

Overview of Schema Matching

Rahm and Bernstein [94, 18] present one of the most comprehensive reviews of the early schema matching approaches. They identify the application fields of schema matching, with most notable applications in *schema integration, data warehousing, e-commerce* and *semantic query processing*. A common task usually tackled is the alignment of schema elements from a source schema to a target schema.

From [94, 18] a common ground from schema matching approaches is their dependency towards the *source* and *target* schemas for alignment, and the underlying instances. This further influences the choice of *matchers* that are set in place to determine whether two schema elements are equivalent, to the nature of the matching approach, namely *instance* or *schema* based.

We guide the analysis of the related work on schema matching and position the work in this thesis, specifically, Chapter 4 based on the taxonomy proposed in [94]. Figure 2.9 shows the taxonomy of schema matching approaches. The proposed schema matching approach for relation alignment in Chapter 4 belongs to the group of *instance-based* approaches. Similar is the matching approach we use in Chapter 3.

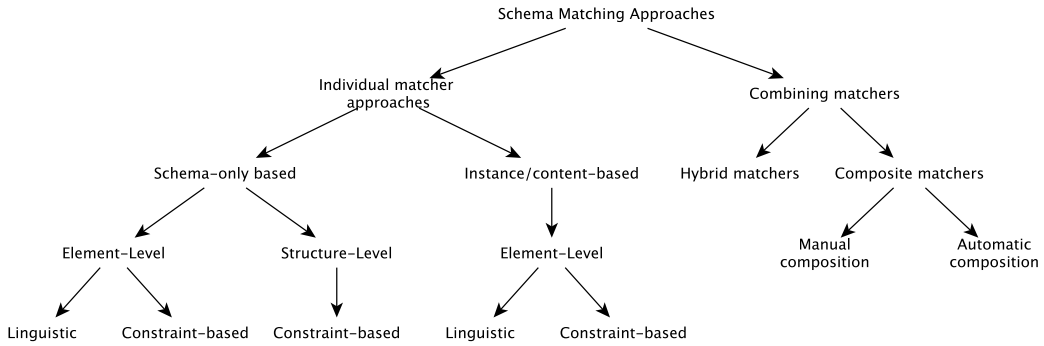


Figure 2.9: A classification of schema matching approaches proposed in [94].

Overview of Ontology Matching

Nowadays, apart from the traditional schema matching that is done in relational and XML databases, a particular subfiled that is closely related to the work in this thesis is *ontology matching*. This is mostly attributed to the increase of availability of Web Data, especially Linked Data. In [99] it is important to note some significant differences between ontology matching and traditional schema matching. The definition of ontology matching is that of matching classes or properties from any two ontologies (see Figure 2.10). Work in the ontology matching field is closely related to the contributions in Chapter 3 and Chapter 4.

Furthermore, the notion of *alignment* has different interpretations. The authors of the survey in [99] have encountered that alignment may stand for one of the four relationships: *subsumption, equivalence, disjointness, more general*, although for a vast majority of works, alignment means the discovery of *equivalence* relationships. In our work, we aim at discovering relationships of *subsumptions* between relations. For us, the relationship of equivalence may be expressed as the two-way subsumption of two relations.

In this thesis, we address two problems that are related to ontology matching approaches. First, we address the problem of aligning output from Web services into a global RDF schema. Second, we propose an *online relation alignment approach* for knowledge bases, where for any two relations we find *subsumption* alignments in the 4-uple definition in [99].

2.4.2 SOFYA: Related Literature

The approach in *SOFYA* performs *ontological relation alignment* where for a given source relation we find target relations in which our source relation is subsumed. Therefore, we review related literature on schema mapping approaches, namely *schema-based* and *instance-based*.

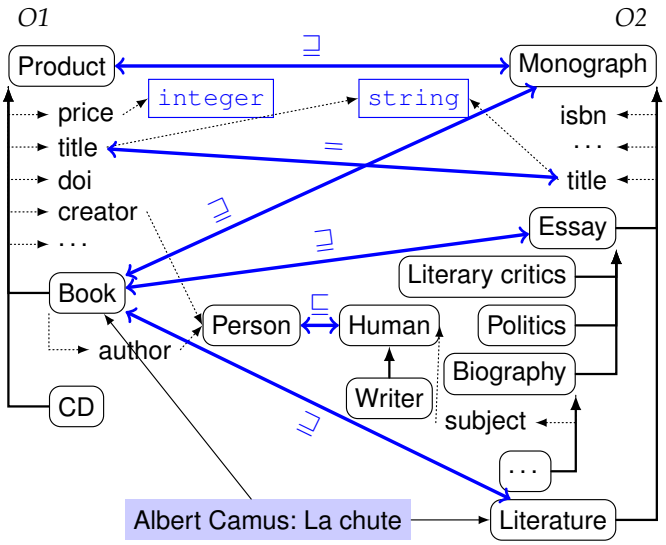


Figure 2.10: Ontology Matching example between two ontologies. The matching problem as the 4-uple, $\langle id, e_1, e_2, r \rangle$. Where e_1 and e_2 are any two classes or properties, whereas r represents the *typed alignment*, that is, *equivalence, disjointness, more general, subsumption* holding between e_1 and e_2.

Schema-based Approaches

From the taxonomy in Figure 2.9, we see that there are three main distinctions in schema matching approaches. In the first group that of *schema based*, the approaches make use only of information gathered from the schema elements (e.g. an attribute and attribute properties from a table). We review work on *schema-based* approaches as they can be seen as complementary approaches towards our proposed approach in *SOFYA*. However, as it will become evident throughout the comparison with existing work, their applicability in our use-case in Chapter 4 has short-comings which we address.

A non-exhaustive list of schema-based approaches is: [68, 70, 69, 35, 36, 77, 76, 82, 84, 83, 26, 74]. Some common techniques used for schema matching in the aforementioned approaches consider the following: (i) constraints enforced by a definition of a schema element, (ii) lexical similarity based on the schema element names (through synonyms, hypernyms, or dictionaries), and (iii) graph-based similarity measures like neighborhood similarity measures between two schema elements in disparate schemas.

Below we present a detailed comparison of *schema-based* related approaches with respect to the contributions in this thesis.

COMA++ [14] is a successor of COMA [34] prototype. COMA++ is one of the most well-known schema matching systems, and it can be safely assumed it is representative of many other schema-based approaches due to its choice of *matchers*. The approach in COMA++ combines several *matchers* (or *matching algorithms*) in order to produce the mapping between two given schemas. Furthermore, COMA++ supports schemas and ontologies written in different languages like SQL, XSD, and OWL. The mapping process

follows three main steps described below.

1. *Component Identification.* In this step it is determined the type of schema components for matching, i.e., nodes, paths, child nodes.
2. *Matcher Execution.* Here, multiple matchers are executed in order to compute the similarity between schema components. There are more than 15 matchers, who consider mostly linguistic and structural similarities. A recent matcher added in COMA++ is an instance based matcher (in support of OWL ontologies) and produces mappings between concepts/classes.
3. *Similarity Combination.* The similarities computed from the different matchers are combined into a unified model to derive correspondences between schema components.

In contrast to our work, COMA++ is a schema based approach for ontology alignment. It focuses on matching ontological concepts (classes in RDF datasets) between two ontologies. The main difference between this work and the contributions in our approach *SOFYA* are the following. Contrary to COMA++, in *SOFYA* we consider ontological relation alignment, furthermore, our approach is *instance-based*. Despite that, the nature of relation alignment is quite different when compared to classes/concept alignment in two schemas. A relation in knowledge bases is characterized by its *domain* and *range* (allowed concepts and literals to be assigned to a relation), hence, an alignment with schema-based approaches is unlikely to work due to the fact that often concepts belong to different taxonomies and follow different naming conventions.

In our case, the objective is to find if a relation is *subsumed* into a target relation, matchers need to determine the subsumption at the instance level, namely a match in terms of *domain* and *range*. Additionally, an argument of the applicability of schema-based approaches could be the alignment of the concepts from the *domain* and *range* of relations. Yet, this goes beyond any existing schema-based approach and can be regarded as a future and different direction to explore for relation alignment where both the domain and range are concepts.

Peukert et al. [88] propose an automated schema matching framework that for a given mapping problem, automatically tune the different *matchers* and order of executed operators in a matching process. The framework mimics the use of schema matching frameworks by expert users or database administrators.

In order to automatically tune the framework for a given mapping problem, the authors in [88] do the following. For a given set of *matching* operators, and any given pair of *source* and *target* schemas, they compute two groups of features that are used to construct rules to guide the mapping process. In the first feature group they compute *schema-level* features, such as data type information of a specific concept in a schema, frequency of elements for a concept etc. In the second group, they analyze the intermediate results that are produced by the matching operators. Finally, from the computed features a set of *matching rules* are constructed. A *matching rule* which represent *expert knowledge* has several attributes such as, where it can be applied in the mapping process (selection, aggregation, mapping etc.), the action it will undertake, i.e. addition of two similarity

matrices produced by different matchers, and finally, it has a relevance score indicating the usefulness of a such a rule.

The rules are mainly of three types, *starting rules* (applied usually in the beginning of a process), *aggregation rules* (combine the output of different matchers), *rewrite rules* (e.g. change the order on how the operators are executed), or *selection* which picks the most probable schema-elements for alignment.

In summary, the introduction of the different features, and the matching rules has the advantage on self-tunning matching algorithms w.r.t to a given mapping problem. In comparison to our system *SOFYA*, the differences are in that we perform relation alignment (contrary to class alignment) with features computed from instances extracted from relations under comparison. An alignment in our case is considered relevant iff a target relation in a schema subsumes the source relation. Furthermore, the approach in *SOFYA* is a supervised machine learning model, which automatically adapts to the different knowledge base pairs by selecting the optimal subset of features, and as such it is related to the selection of *matchers* in related work.

Similarly, Albagli et al. [11] propose the *iMatch* schema matching approach; a system that produces matches between two given ontologies by using the structure of the ontologies and the type information between classes. *iMatch* produces matches between classes through models which represent Markov Networks [87]. In this work the authors do not tackle the relations matching problem, which is the focus of our work in Chapter 4.

Doan et al. [35] proposes a schema matching approach where the source data is in XML and the source and target schemas are in DTD. The approach is supervised, where the user of the tool provides manual mappings between the two schemas. From the provided mappings, the systems learns several *base learners* (based on different matchers) between the two equivalent concepts in the two schemas, and finally combine them into a *meta-learner* that predicts the relevance of a mapping. Furthermore, to avoid incorrect mappings, the user has to add examples of what should not be aligned, e.g. *house-ID* should not mapped with a *num-beds*.

In contrast to our ontological relation alignment, *SOFYA* in Chapter 4, were we consider similarly a supervised approach, the differences are the following. In our approach in *SOFYA*, value distributions or expected values from a specific attribute in a schema do not apply as we consider entity relations. Finally, such an approach is not applicable in our setting as it requires users to provide exclusion rules from the mapping process that prohibit two specific attributes to be aligned in disparate schemas.

Madhavan et al. [73] propose a schema matching approach that use pre-aligned XML schemas to estimate what are possible matches between the different schema elements. Finally, they propose only possible matches between classes/concepts for any two schemas, however, they do not consider the actual mapping.

Instance-based Approaches

The second major category of schema matching approaches are *instance-based*. They rely on actual instances from the corresponding schemas. The difference here lies that approaches not only use the schema information and schema element definitions, but also consider instances from them. The advantages over *schema-based* approaches is that when the schema definitions provide limited information for the different matchers set in place to perform the alignment. Hence, through the instances one can analyze for a specific schema element the distribution of values, the data types etc., and accordingly perform a mapping between schema elements.

Approaches like in [68, 70, 69, 35, 36] rely on textual similarity from the actual instances, or use the distribution of possible values from a schema element, correspondingly from its instances to compute the mapping between two schemas. Below we compare individual *instance-based* approaches to our approach in *SOFYA*.

Doan et al. [37] propose *GLUE* a semi-automated machine learning framework for the ontology mapping problem. The mapping problem is defined as matching a node from one ontology into the node of another ontology. The mapping is done only for the case of one-to-one mappings. GLUE has three main components.

For two ontologies U_1 and U_2 and concepts A and B, learn a classifier for A as following. Instances that exist in A are labeled as positive instances, and those that exist in B (but not in A) as negative. Then the classifier is applied to concept B, and labels the individual instances. The features are extracted from instance attributes, such as the name or the content. Finally, the mapping between A and B is done under the following assumption *'if all children of a node X match node Y, then X also matches Y'*. That is, in the case where instances from B are classified mostly as positive instances from a classifier learned for A, then the corresponding concepts should be aligned.

In contrast to our work, GLUE focuses on matching ontological concepts (classes in RDF datasets) between two ontologies, while in our case as part of *SOFYA* we discover relations alignments. Furthermore, such a model proposed by GLUE would not be applicable in our case, as we would need to train classifiers for the individual relations (step (1) and (2)). This presents a major drawback in applying it for relation alignment, where such models would be highly likely to overfit to the specific relations.

Wang [124] et al. propose an instance based ontology matching approach. The approach uses machine learning to distinguish between correct mappings of any given pair of concepts/classes from two ontologies. The features are computed from the instances that belong to corresponding concepts. The features are mainly of two groups. The first group is represented as a bag-of-words with terms coming from the textual literals from the instances and the corresponding frequency of each term when aggregated at the concept level. The second group, considers features extracted from the metadata from the concepts.

Finally, a classifier a Markov Random Field, is trained on the computed cosine similarity on the bag-of-words extracted from the literals and metadata.

The approach is evaluated on very limited collections of digitized books collection, and on a collection of multimedia documents. The differences with our work in Chapter 4, namely the system *SOFYA*, is that we consider ontological relation alignment. Furthermore, such an approach is tailored to its specific domain of book collections, where bag-of-words similarities are likely to work. Unlike in our case where we consider *entity relations* where the amount of text that can be used is insufficient.

Qian [92] propose a sample-driven schema mapping. In the approach, the users play a central role by providing sample instances which are then used to find possible mappings to target databases. The process is iterative, in the way that the more samples are provided by the users, the more accurate are the produced mappings.

It must be noted here that the scenario is somewhat different from traditional schema mappings approaches between a given source and target schema. In this case, the user wants to aggregate multiple databases or schemas into a single one (the target schema). Hence, without formally specifying the attributes of the target schema but instead providing only sample instances the mappings are generated w.r.t the source schemas.

The approach in [92] is similar to our work in Chapter 4 only at one point. We sample for entity instances that are associated with a relation, which we want to align in a target knowledge base. Dependent on the sample size, our approach will be affected in terms of coverage. On the other hand in [92], the impact is demonstrated at the attribute level that can be mapped to a target schema given the provided samples. Yet, we are different in the sense that our approach extracts the samples automatically, and that we consider relation alignment, instead of class alignments.

Similar to the work by Qian [92], Miller et al. [75] rely on samples on how specific values in source schema should be mapped to a target schema. The samples are provided by database administrators, and they show how a mapping in a target schema is constructed, e.g., *salary = PayRate * WorkHours*.

They follow a strategy where they construct correspondences showing how a value from a target database can be constructed from the executed queries on the source database. However, such an approach proposed in [75] requires samples about how example values from a schema should be mapped into a target schema added by a database administrator.

In the case of *SOFYA* we have significant differences in that we consider relation alignment, where such correspondences in [75] do not apply. Furthermore, our approach is performed in an online setting without any intervention from a user.

In ROSA [46], the authors consider similar to the work in this thesis, namely Chapter 4, the relation alignment problem. However, in this thesis, we consider the proposed approaches in [46] as our baseline, and further propose new measures for relation alignment. The pca measure used in [46] represents one of the features of our supervised learning module. Our experiments on 6 cases of KB alignments have showed that our approach outperforms significantly the proposed approach in [46]. On average, for all 6 cases of KB alignments, in terms of precision we achieve a gain of more than 55%, whereas in terms of F1 measure, with 41%. A detailed comparison is shown in Table 4.4.

Galárraga et al. [45] propose an approach to map OpenIE knowledge bases like NELL [25] to structured knowledge bases like Freebase. The approach consists of two steps: (i) mapping entity mentions in facts from NELL into the equivalent entity instances in Freebase, and (ii) mapping the predicates or verbal phrases from a fact in NELL into relations in Freebase. In more details, our approach of ontological relation alignment in SOFYA in Chapter 4 is related only to the second step in [45].

In [45], verbal phrases from a fact in the NELL KB are clustered together based on the computed equivalences based on the AMIE approach [47]. In this case a prerequisite is that the subject/object entity mentions in a fact in NELL are already resolved to Freebase entity instances. Next, the verbal phrases within a cluster are canonicalized into one of the Freebase relations [46] based on the common entities that serve as subject/object in fact in the NELL KB, and respectively in Freebase.

In contrast to [45], in this thesis, specifically in the SOFYA approach we deal with ontological relations, where relations within a KB are considered to be unique. Whereas for the mapping of ontological relations across disparate KBs we use the approach in AMIE as our baseline and address its major shortcomings explained in Chapter 4.

In [54] authors suggest a rule learning approach for ontology matching. The rules are learned by means of analyzing instances of a given pair of ontologies. Similar to our approach proposed in *SOFYA*, they limit the set of instances for the learning process, to instances that have identical URIs or are linked with `owl:sameAs` for an ontology pair.

In the ontology matching process they distinguish between two main use cases. In the first use case, they perform 1:1 mappings between concepts/classes. The features used to learn the rules conform to boolean indicators w.r.t the presence of a particular `rdf:type` (which represent classes or concepts in an ontology) for an instance. The instances are represented into a high dimensional feature space, where each instance has 98k features. The features corresponds to the attribute-value pairs extracted from the instance. Finally, the rule learning process follows the intuition that classes that co-occur in a large set of instances are likely to be aligned.

In the second use case, the matching process considers relations from an ontology pair. Here, the rule learners analyze all instances that have a relation that are not mapped to a target ontology, and the features correspond to attribute-value pairs. For each relation a separate ruleset is learned. The mapping of relations in two disparate ontologies is done by analyzing the similarity of the rulesets. Rulesets that are similar indicate that two relations are equivalent.

In this thesis we similarly tackle the problem of relation alignment. The differences lie in the fact that we consider a more realistic scenario where the relations are not likely to be equivalent but rather one subsumes the other. We note this especially in the considered datasets in [54], namely YAGO and DBpedia. In which case, relations from DBpedia are more likely to be subsumed by their counterparts in YAGO. Furthermore, we perform the alignment process during query-execution and operate with a minimal amount of instances from the ontologies. Additional advantages of our approach is the minimal overhead in query-execution, hence, making it a realistic framework that can be applied in an online setting.

Finally, contrary to [54] where one needs to find the similarity of rulesets in order to align to relations, in our setting we perform this in automated manner. Furthermore, if one considers as features the actual attribute-value pairs, the learned rulesets are specific to an ontology and relation pair, hence, the need to re-learn such rules. Contrary, we compute features that generalize well across relations and ontologies.

In [105] authors propose a system called *ONARM*. It is an ontology mapping technique where given two ontologies is able to map *concepts* (classes) of these two ontologies by extracting association rules. The proposed approach exploits the structure of the input ontologies and more specifically the hierarchy of concepts in order to determine the mapping between them. It applies linguistic similarity on the labels of concepts along with a structural similarity technique.

Summary

Bernstein et al. [18] provide a thorough overview of schema matching approaches, the applications, and future applications of schema matching on the web. It is noteworthy, that our use of schema matching on mapping Web service output into a global schema and the use of relation alignment online through minimal samples of instances from a given pair of datasets present real-world use cases and emphasize the importance of schema matching on the Web and Web data in general. This is inline with their view of schema matching approaches and their use cases.

We depict two main factors for ontology matching that apply in our relation alignment problem in Chapter 4. The first factor deals with the choice of *matchers* that are used in the ontology matching task, which is dependent on the datasets under consideration. In our work, we perform feature selection for our machine learning model for the alignment task, hence, choosing the appropriate matchers for which our alignment model has optimal performance. The second factor that we see as common point in [99] and our work, is the choice of training individual classifiers for the different datasets under consideration. That is to say that for any pair, alignment models might be generalizable and work for any arbitrary pair, however, optimal performance is achieved when the models are fine tuned w.r.t the datasets under consideration.

Finally, the discussed related work focus on mappings between classes in ontologies, whereas in our work we consider subsumption alignment between relations. Furthermore, one major drawback and difference with our approach in *SOFYA* is that related work requires full-access of the datasets under considerations. In our case, we propose an efficient approach that performs the alignment in an online setting with minimal samples from the datasets. Lastly, through relation alignment we enable end-users to access the wealth of LOD through query-rewriting and without requiring pre-aligned datasets at the class/concept level.

2.4.3 DORIS: Related Literature

In this subsection, we review related work that is closely related to our approach *DORIS* on mapping the output of Web services into a global schema in Chapter 3. We organize the related work into the different topics that deal with: *discovery*, *representation*, *orchestration* and *wrapper induction*. Below we describe a related work that is closely related to our contribution in *DORIS*.

Schema Matching for Web Services

Determining the description of a Web service in terms of a global schema can be seen as a special case of schema mapping or ontology alignment. This is the problem of matching the concepts and the relations of one schema to the concepts and relations of another schema. Several approaches have been developed to this end.

Similar to our work DORIS, Karma [104] proposes to integrate REST services into a target RDF KB given as input, by describing them in terms of the target schema. Unlike DORIS, Karma assumes that there is an obvious translation of the XML call results into tables. The challenge in Karma is to assign semantic types to data columns. To this end, a supervised machine learning algorithm based on the conditional random field (CRF) model [65] is proposed. By design, user intervention is required during the mapping process. The user might be asked to specify the appropriate semantic type for some columns. The CRF model is then re-trained using the new corrected types assigned by the user. Unlike Karma, DORIS does not rely on a standard translation of the call results into tables. Actually, one of the challenges that we face is to discover how the descriptions of the different types of entities returned by the service are nested in the call results. Furthermore, our approach is fully-automatic and it does not require the intervention of the user during the mapping process.

Several approaches use the overlap of instances to compute the alignment between two KBs [102, 55]. For us, the instances of the one "KB" is a tree (the call result of the WS) in which some nodes may correspond to entities, and other nodes are just intermediate nodes. The tree does not tell us which nodes are entities and which are not. One solution is to convert every possible path of the tree into a fact. However, as we will see in Section 3.6, this solution does not work well for Web services. The reason is twofold: First, a node in the tree may actually combine the properties of several entities, which leads the approach ad absurdum. Second, a fact from the call result may correspond to a join of several relations in the KB. State-of-the-art approaches for KB alignment, however, expect a one-to-one mapping.

Works such as [98, 123, 52, 128] aim to bootstrap a global schema. In [98] e.g., the authors show how the schema of a new ontology can be bootstrapped by using as input

the WSDL schema [3] descriptions of the Web Services using the SOAP protocol [4]. In our scenario, we target the REST protocol, which provides no such information. Rather, our goal is to map services into an existing schema.

Delobel et al.[32] map XML data sources into a uniform schema. The mappings from the XML data sources into the mediated schema are based on semantic and structural similarities. However, here the difference to our work is the mapping is carried from XML to XML, whereas we map the output of Web services into a global RDF schema.

Related work [71, 107] on a slightly different setup, where HTML Web tables are further enriched with semantics, e.g. columns in a Web tables are associated to entity types in existing knowledge bases, and pairs of columns are associated to relations.

In our case, contrary to Web tables, we deal with XML documents, namely XML paths which are our target of associating them to entities or relations in a target knowledge base schema. The main issue here is the different nature of our data, where the data is a tree and there are multiple paths from the root to the leaf nodes, and such approaches for annotations are not applicable. For example, questions that arise are what is the depth level that we should consider to map paths to relations? Furthermore, in some cases at specific depths we map the path to an entity rather than to a relation in a KB.

Query Discovery

In Section 2.4.2 we presented a review of the work by Miller et al. [75] on a schema matching approach that relies on queries from a source database to map in target schema, and furthermore uses query constraints to construct correspondences between a target and source database. Specifically, they show how a value from a target database can be constructed from the executed queries on the source database. Such an approach requires samples about how example values from a schema should be mapped into a target schema which are added by an database administrator.

In this thesis our schema matching approaches differ significantly. For our system *DORIS*, the schema matching is considered under the scenario where we have only one KB which represents at the same time our target schema, and a Web service which has no formal schema. Even after we call the service, we do not necessarily get information about the schema of the Web service. This is because the output of the service usually contains only vacuous node labels and no concept names. Therefore, schema based approaches cannot be applied in our setting.

A closely related work to *DORIS* is [126]. It addresses the problem of transforming XML data to RDF triples that conform to a target schema. However, they adopt a Global-as-View approach where the schema is expressed in terms of the input data sources, while in our approach we construct a Local-as-View approach, where we have a global schema and the data sources, in this case, output from Web services is mapped accordingly.

3. `http://www.w3.org/TR/wsdl`
4. `http://www.w3.org/TR/soap/`

Wrapper induction

Wrapper induction algorithms [33, 44, 93, 122] are concerned with automatically constructing information extraction rules ("wrappers") for Web pages. However, wrapper induction does not map the sources to a global target schema. Our work, in contrast, translates the schema of the WS into the classes and the relations of the KB.

Web Service Representation

Following the vision of Semantic Web, several approaches have been proposed for describing services using semantic formalisms. Notable approaches like WSDL-S [113], OWL-S [110] (the successor of DAML-S [67, 86]), and WSMO [112] are already W3C standards. The first standard, WSDL-S [113], proposes semantically enriched WSDL documents for SOAP Web services. In contrast, the Web services that we address in this work do not expose WSDL descriptions for their input and output types. The other two standards, WSMO [112] and OWL-S [110] share the purpose of representing semantic information for Web services in order to enable automatic service discovery, composition and execution. Both specifications are based on description logic languages. In contrast, the Web service descriptions that we aim at computing are views with binding patterns [95], a special case of Datalog rules [8]. The two languages are not equivalent. In [23], it is shown that the Description Logic language DL-Lite can be embedded in an extension of the Datalog language. The major extension allows existentially qualified variables in the heads of the rules. Furthermore, we are not aware of systems that automatically derive Web service descriptions expressed in the above description logic languages.

In [85] authors propose a framework that provides descriptions of REST Web services with main goal the Web service discovery. In this work WSDL files are not required. They search for extra information concerning a Web service in provider documentation Web pages (e.g. *ProgrammableWeb.com*) or in third-party Web sources like wikis or Web APIs repositories. They introduce semantics in the final description of the service by extracting ontological entities from text by using *named-entity recognition*(NER) tools [79]. The NER process is performed on the properties of a Web service which contains values like the *description, operations* etc.

A recent survey [108] summarizes the different means to describing Web services. Similar to our understanding, in this survey, the authors emphasize the rationale of the several Web Service descriptions approaches, mainly attributed to the limited take up of standards like WSDL and WADL. The argumentations goes inline with the widespread of micro-formats for describing structured data on the Web.

It is worth noting that in majority of the cases the descriptions are provided mainly to the operations and the corresponding input and outputs provided by a Web Service. For example, hRESTS [58] provides an annotated description through micro-formats such as XHTML for a description of an Web Service provided in natural language. In [58] they focus on describing the allowed operations, the input parameters, and the output format.

Speiser and Harth [101] propose an Linked Data Services (LIDS) to describe formally through RDF and SPARQL the input and description of a Web Service. The input parameters of a Web Service operation are binded to specific ontologies. Similarly the output of a Web Service is mapped to a triple pattern in SPARQL with variables linked to an ontology. These so called LIDS descriptions for the input and output of an operation, however need to be generated manually on a case by case basis. This provides a major drawback of LIDS and this is not explained in [101] how one could further generalize this into an approach that can be applied to any Web Service.

Similarly, Krummenacher et al.[64] propose a conceptual model for describing Web services as Linked Open Services (LOS). For an LOS the input and output of an operation are described as RDF graphs. Specifically, the input parameters are RDF triples, consequentially the output of LOS is an RDF graph. Being a conceptual model, this proposal fails to address the problems with existing Web services, which in majority of cases do not provide well formed descriptions. From a statistics provided in [41], nearly 44% of Web services communicate in JSON which does not imply any structured description of input parameters nor output formats.

Summary. In summary, related research on Web service representation [85, 108, 58, 101, 64] has focused on providing representations mainly for the operations and their respective parameters. Furthermore, in many cases, the representations is not structured.

In contrast, the approaches we propose in *DORIS* provide a global RDF schema for the output of any Web Service without requiring any additional information. Hence, allowing to combine output from multiple Web services seamlessly due to the global RDF schema, and hence infer additional knowledge.

Orchestration and Mash-ups

Web service orchestration is concerned with joining several Web services in order to reply to a query [15, 91, 16, 90]. This work does not map services to a common schema. On the contrary, it requires that mapping as input. Other work proposes a new semantic model for representing Web services or mash-ups [89, 106]. Our work, in contrast, aims to represent Web services in the standard model of views with binding patterns [50].

Web Service Discovery

The process of finding suitable Web services for a given task is called *Web service Discovery*. For this purpose, the Web services are automatically annotated with the concepts for which they can return instances.

A typical scenario in Web service discovery is the following: A user provides the description or definition of the Web service that she would like to use along with the query that she wants to answer using the Web service to a repository of existing Web services. As a result the corresponding system identifies the most suitable Web service to user's need. There has been extensive research in Web service discovery, however, in this section we focus on the most important works.

In [40] authors propose a system called *Woogle* which focuses on similarity search for Web service operations. Woogle works with SOAP Web services whose associated WSDL files are accessible and contain information about their functionality description, inputs and outputs parameters. Woogle computes similarity between different Web services by combining multiple sources of evidences. As evidences they use the score obtained after the application of traditional string similarity techniques(e.g. *bag of words* similarity) to the names of Input/Output parameters, and to the description of the Web service. Apart from the traditional string similarity they make an other step. They propose clustering algorithm that groups names of parameters (inputs and outputs) into semantically meaningful concepts. These clusters are used later on to determine the similarity between the Input/Output parameters.
The different types of similarities are combined using a linear function, with weights assigned manually after the experimental evaluation.

In [100] authors proposed an approach of ranking the top-k most relevant Web services according to user's need under multi-criteria matching.

In this work authors assume that the description of a Web service is already given in a *WSDL* file. They proceed in two steps. First, they use a matchmaker, similar to OWLS-MX [56] in order to match the Input-Output (I/O) parameters of a service coming from the WSDL file, to user's request. The matching is made by exploiting different techniques like logic-based reasoning, bag-of-words similarity, or Jensen-Shannon information divergence. For a request R and a similarity measure, the system calculates the matching score between the parameters in R and parameters in candidate Web services (S), which is used to generate a top-k list of ranked services.

Cardoso [24] propose an approach to find matching Web services for a given Web service request. In the proposed scenario, the *request* and the *advertisement* Web services are already pre-described with ontologies. Namely, their input, output, operations, requirements, etc., specifically the concepts used to describe these operations are already aligned to an ontology. Furthermore, they distinguish two main use cases. In the first use case, the descriptions from the request and advertisement Web services come from the same ontology, and in the second use case they are from different ontologies.

The matching of a request Web service to a target advertisement Web service is done by considering the overlap in terms of *properties* from the concepts that are in common between the two, where the properties correspond to attributes pre-defined in the ontology. In the second use case, as similar concepts are considered those that have a string similarity (e.g. Levenshtein) over a given threshold β.

Alacron et al.[10] propose an approach for Web Service discovery by distinguishing between two main attributes of a Web Service. The first attribute is the so called *semantic layer* which captures the semantics of the parameters, resources, and the actions of a Web Service, and the *activity layer* that models the interaction between the different components of a Web Service.

The descriptions are provided as Microdata or Micorformats, allowing a smooth integration of the description into the Web pages where these services reside. The discovery process assumes that the user is aware of the schemas used to represents the descriptions (in their case *schema.org*), and allows them to query for different functionalities in SQL like syntax. The output can be any of the defined concepts in the *semantic layer*.

A disadvantage here is that the users are supposed to be aware of the concepts in *schema.org* and manually find operations that allow them to do a certain task. In comparison to our contribution, this work does not provide any information regarding the possible schema of the produced output by a Web Service operation.

Summary. In contrast to our contribution in *DORIS* in Chapter 3, work in Web service discovery differs significantly. In our case we are interested in describing the output of a Web service in a global RDF schema. Furthermore, we compute views with binding patterns that map the entire call result to the target knowledge base. A fundamental difference is that the works in [40, 24, 100] either assume existence of already formal descriptions of the Web services in the form of ontologies or WSDL, or lack experimental evaluation of the proposed approaches. Finally, some of the proposed approaches, e.g. [10] are complementary to our work in *DORIS* and would further help users to find services.

Chapter 3

DORIS: Discovering Ontological Relations In Services

In this chapter we present DORIS (Discovering Ontological Relations In Services). The approach in *DORIS* deals with the problem of mapping the output from Web service operation calls into a global RDF schema.

The importance of mapping the output of Web service operations is further supported by the large number of Web services that are exposed as REST APIs [41]. REST APIs do not provide access to the schema that is used to represent such output, even though, such a schema might be explicitly stated in its source. Therefore, such problem is seen as a major hindrance towards the full integration of Web service, and furthermore to uniformly interpret the output from different operations and services.

We propose the approach *DORIS*, where we construct mappings from Web service operation calls to a global RDF schema. We present new algorithms for inferring the *view definitions* of Web service in this global schema automatically. We also show how to compute *transformation functions* that can transform Web service call results into the global schema. The key idea of our approach is to exploit the intersection of Web service call results with instances from a knowledge base and other call results.

The rest of this section is structured as follows: Section 3.1 describes the problem that this chapter tackles. Section 3.2 lists the assumptions we make after a thorough analysis of the Web service landscape. Section 3.3 presents all the steps of our algorithm about discovering Web service's schema using a global schema. Section 3.4 shows a baseline approach to our subject. Section 3.5 presents the prototype of our DORIS systems. Section 3.6 presents the experimental evaluation of our approach. Finally, Section 3.7 describes our algorithm for discovering I/O dependencies between Web services, before Section 3.8 concludes our findings.

3.1 Problem Description

Modeling Web Services. A Web service is essentially a parameterized query over a remote source. In this work, we concentrate on REST Web services, motivated by the fact that more than 80% [41] of Web services use the REST architecture (see Section

Web service url:
`http://www.last.fm/api/getAlbums?query=artist`
Parameters:
artist (Required): The artist name

Figure 3.1: A REST Web service

2.2.1). These services work by accessing a parameterized URL. For example, *Last.fm* offers a Web service which, given an artist as input entity, returns the birth date, the gender, and the albums of the artist. Figure 3.1 shows how this service is presented on the *Last.fm* Web page. We say that *artist* is the *input type* of this Web service. In other words, we use the term *input type* to refer to the class of real entities that a Web service expect as input in order to return valid and meaningful output. Typical input types may include `Artist`, `Writer`, `City`, `Movie` etc. To *call* the Web service of Figure 3.1, the user has to replace the string "artist" by an *input entity*, e.g. "Frank Sinatra" and access the URL. The server responds by sending the results of the request, which are usually in a semi-structured format like XML or JSON. Figure 3.2 shows the result of our example call. It contains the albums of "Frank Sinatra" with their title, release date, an internal identifier given by the Web service provider and all their singers, including Sinatra himself. For each singer, apart from the name, the results include the gender, the birth date, and death date in case the singer has passed away. The challenge is now to make this output interoperable with the results from other Web services.

Global Schemas. In order to make the service interoperable with other services, the state of art solution [49] is to assume a common global schema. Then, the Web services are represented as parameterized conjunctive queries (i.e., views with binding patterns) over this global schema. In this work the role of the global schema represents a schema from existing well-known knowledge bases. Knowledge bases such as YAGO [103] or DBpedia [13] provide a schema, a taxonomy of classes with their instances and millions of facts in that schema. The instances can be used to probe the Web service. The facts can be used to guess the schema of the Web service. An example of such a knowledge base is depicted in Figure 3.3. Here the global schema contains relations such:

- *created*: for an artist who released an album;
- *label*: for the relationship between an entity and its name;
- *birthdate*: for the relationship between an artist and her birth date;
- *hasChild*: for the parent-child relationship between two entities;
- *date*: for the release date of an album;
- *diedOnDate*: for the death date of an artist;
- *gender*: for the relationship between an entity and its gender entity.

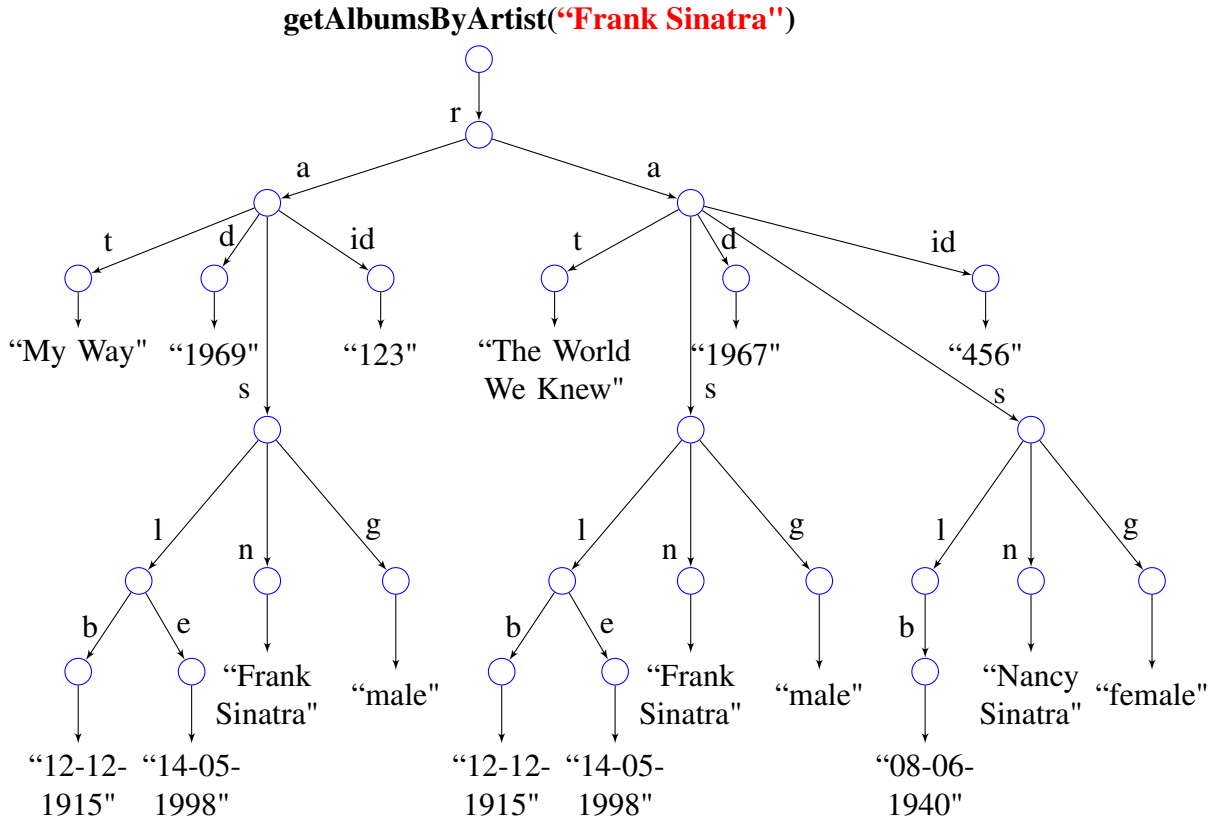


Figure 3.2: A call result for getAlbumsByArtist with input value *"Frank Sinatra"*

Expressed in this global schema, the service becomes the query shown in Figure 3.4. Such a view is essentially a *parameterized conjunctive query* as was described in section 2.3. Every variable in the head of the view is adorned with i, for input-output variables or o, for output variables respectively. More precisely, y_4 is the *input parameter* (the name of the artist) and all the other arguments are *output parameters* (the birth date, the gender, the album, and the release date of the album). It is common to distinguish between an entity and its name. For example, while the g will be some internal identifier for a gender, the y_5 will be the label of that gender, such as, e.g., "female". In this view definition, the result of our Web service becomes a set of tuples (Figure 3.5).

The goal is to have all Web services expressed as views over the same global schema. In this way, applications can uniformly reason about Web services in the terms of that common global schema.

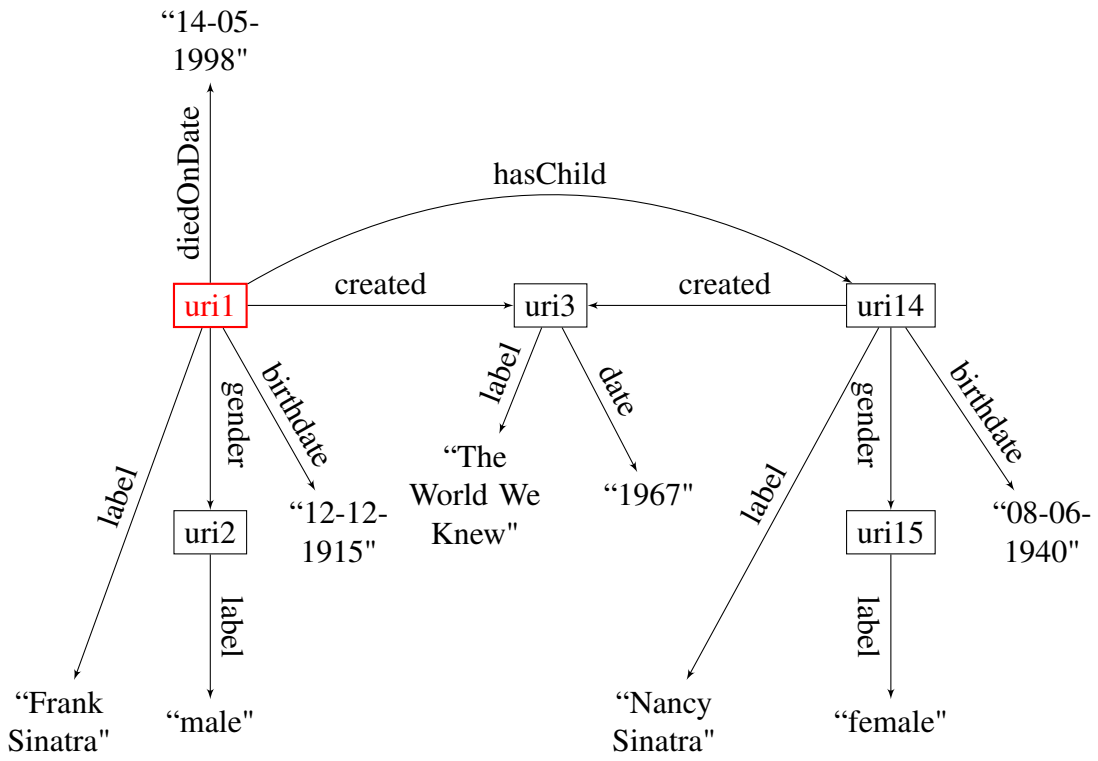


Figure 3.3: RDF (fragment).

$$getAlbumsByArtist^{iooooo}(y_4, y_1, y_2, y_3, y_5, y_6) \leftarrow label(x, y_4), birthdate(x, y_3), gender(x, g), label(g, y_5), created(x, z), label(z, y_1), date(z, y_2), diedOnDate(x, y_6)$$

Figure 3.4: View with binding patterns for the Web service getAlbumsByArtist.

(Frank Sinatra, My Way, 1969, 12-12-1915, male, 14-05-1998)
(Frank Sinatra, The World We Knew, 1967, 12-12-1915, male, 14-05-1998)
(Nancy Sinatra, The World We Knew, 1967, 8-6-1940, female, -)

Figure 3.5: Call result of getAlbumsByArtist in the global schema

Problem Statement. The central challenge with global schemas is that each Web service has to be mapped into this global schema. While the orchestration of Web services has received much attention lately, the transformation of services into the global schema is often done manually. Our goal is to deduce the schema fully automatically. More

precisely, our goal is the following.

Given:
- the URL and the input type of a Web service (Figure 3.1)
- a global schema (Figure 3.3), where we decide to use existing knowledge bases

Compute:
- the view of the Web service in the global schema (Figure 3.4)

The URL of the Web service and the input type can be found by a non-expert user on the Web page of the Web service (as in Figure 3.1). No knowledge of the output schema of the Web service is required. On the contrary, our goal is to deduce this output schema automatically.

This problem is challenging for several reasons. First, the node labels in the call results are usually vacuous and do not give away any semantics. In the example (Figure 3.2), they are just "r", "a", and "t". Second, it is not clear which nodes in the call result correspond to entities in the global schema. In the example, one can guess that the nodes labeled with "s" correspond to singers. However, the nodes labeled with "l" correspond to nothing in particular. Third, relations in the global schema can correspond to paths of different length in the call result. For example, the gender of an "s" node is 1 hop away, but the birth date is 2 hops away. Finally, the goal is to transform a call result (Figure 3.2) into tuples in the global schema.

3.2 Observations and Assumptions

A Web service returns information about the input entity. The problem is that this information can be expressed in a multitude of ways in the call results. This is because XML and JSON do not define a special syntax for encoding semantic relationships. For example, the labels of the tree nodes do not have to correspond to relationship names, let alone to relationship names from our KB. JSON even allows creating nodes that have no label at all. Furthermore, XML schemata can contain an arbitrary number of intermediate nodes between two entities. For example, the XML tree in Figure 3.2 separates the name of the album ("The World We Knew") by 3 intermediate nodes from the singer of that album (*s*), while the KB in Figure 3.3 uses 2 intermediate nodes. However, we have noticed that in practice, Web service call results follow a number of common sense principles. We will now describe these principles, and use them as assumptions for our work.

I. The KB and the WS overlap. If the Web service and the knowledge base are from the same thematic domain e.g. *music*, or if the knowledge base is a general purpose knowledge base e.g. DBpedia, then we expect to have an overlap between the entities that the knowledge base knows and the entities that the Web service knows.

II. The call result is connected to the input entity. A Web service call always contains data related to its input entities. Hence, we expect that the entities contained in the call result appear in the KB and that they are connected to the input entities by paths that do not exceed a certain length. In practice, this length is not larger than three [81].

III. Mapping from entities to nodes. Whenever a call result contains an entity (of the knowledge base), this entity is usually represented as a structured node with sub-nodes. For example, in Figure 3.2, every album is represented by a subtree that is rooted in an *a*-node. Hence, we assume that for every entity in call result (XML tree), there is at least one structured node that is shared by no other entity of the same class. We call this node an *entity node*.

IV. Relations correspond to linear path queries. We observe that typically, a relation between two entities is encoded as a path between the entity nodes and that the sequence of path labels is used consistently, i.e., it always expresses the same relation.

For instance, the relation *created(*x, y*)* for singers and their albums in the KB corresponds to the WS path */r/a(*x, y*)* in the call result in Figure 3.2. Here, we use x to denote both the root of the call result and the input entity from the KB (Observation II). We use y to denote both the album entities from the KB and the nodes in the result. Similarly, the relation *label(y,z)* for albums and their titles corresponds to the WS path */*τ*(y,z)* at album nodes y.

We note that the same observation applies to the properties of the entities. For instance, the path */r/a/t/*τ selects the titles of the albums released by the input entity x. In our KB, such relationships are expressed as the conjunction of two facts of from *created(*x*,* y*). label(*y*,* z*).*

V. Text nodes map to literals in the KB. We have noticed that text nodes typically correspond to literals in the KB. Using a simple string similarity function, we can map a text node to an equivalent literal in the KB in the vast majority of cases.

3.3 Web service Schema Discovery

3.3.1 Overview

The algorithm requires as input (1) a KB, (2) the URL of the WS, and (3) the input type of the WS. We assume that the input type is given as a class of the KB. Our algorithm proceeds in 4 steps, also sketched in Figure 3.6:

1. **Probing**: We call the WS with several entities from the KB and obtain sample call results (Section 3.3.2).
2. **Path Discovery**: We discover paths in the call results from the root to literals (Section 3.3.3).
3. **Path Alignment**: We align the paths in the call results with paths in the KB (Section 3.3.4).
4. **View & Transformation Function Construction**: Based on the aligned paths, we build the view and the transformation function (Section 3.3.5).

We will now detail these steps.

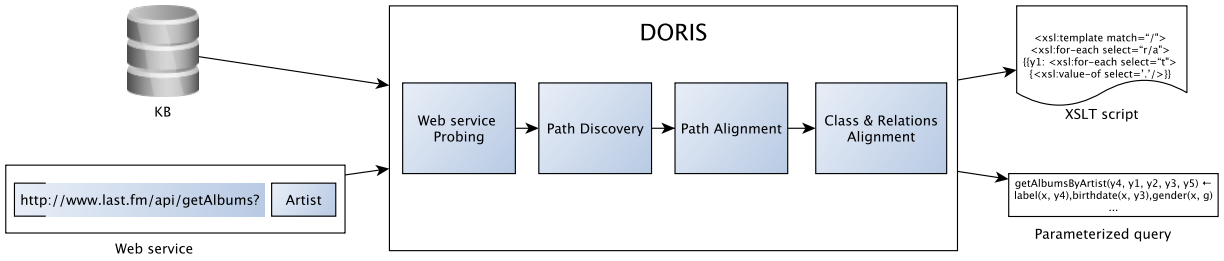


Figure 3.6: Overview of DORIS's processing steps.

3.3.2 Web service Probing

Our schema discovery algorithm requires a number of sample call results from the WS. To generate these, we make use of the input type of the WS: We call the WS with instances of the input type from the KB. For example, if the input type is given as `Singer`, we call the WS with instances of the class `Singer` from the KB.

Not every instance of the input type in the KB will return a valid call result from the WS. The WS may not know all instances. However, it is likely that the WS stores data for known entities such as famous actors, acclaimed books, or big cities. Our intuition is that the KB, on the other hand, will likely contain more facts about such "important" entities (Observation I). Therefore, we rank the entities of the input type by the descending number of facts about them, and call the WS with the top k of them. In our experiments (Section 3.6), we show that a number of $k = 100$ samples is usually sufficient. This corresponds to less than 0.01% of the data that the tested APIs contain.

Some WSs have more than one input. If we encounter a WS with input types $t_1, ..., t_n$, we first find "important" entities of t_1. Then we find in the KB the entities of $t_2, ..., t_n$ that are connected to entity of type t_1. Finally, we probe the WS with these n entities. In what follows, we treat the call results as if they were call results about the single input entity from t_1. This will not hurt our mapping algorithm, since the other input entities are connected in the KB. If they are present in the call results, they will be discovered.

The probing gives us a set of input entities for which the calls have been executed and the results have been materialised. We call such entities *sample entities*.

3.3.3 Path Discovery

Given a sample entity x, our goal is to align paths in the call result that originate at the root $\lambda(x)$ with paths in the KB that originate at the input entity x. For example, for the call result in Figure 3.2, we want to find that the path $r/a/t/\tau$ connects the root to the title of an album. So we want to align the path $r/a/t/\tau$ in the call result with the path $created.label$ in the KB. In the following, we assume a fixed sample input entity x, and we write $(r/a/t/\tau, created.label)$ for our path alignment. To find these path alignments, we first generate all paths in the call result from the root to a literal, and all paths in the KB from the input entity x to a literal, and then align these paths (Observation V).

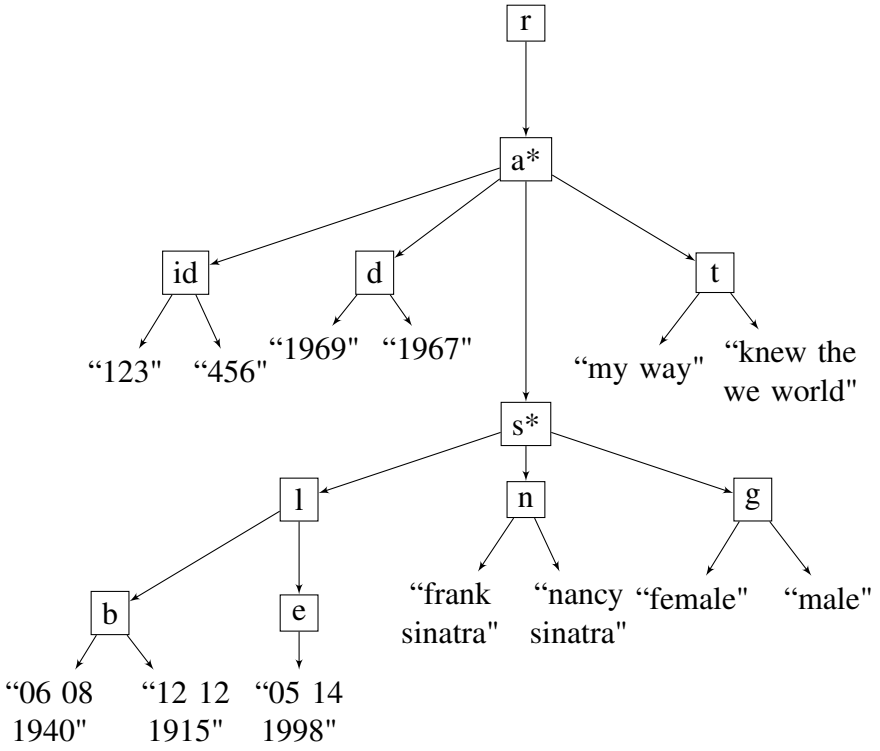


Figure 3.7: DataGuide of a call result.

Generating WS Paths. We first enumerate the paths from the root $\lambda(x)$ to text nodes. The result is a set of pairs of a path p and a text node y, (p, y), *s.t.* $p(\lambda(x), y)$ (Definition 2.2.4). This set is easily computed in a depth first traversal of the XML tree by constructing a summary of its data paths in the form of a DataGuide. A DataGuide, as it is presented in section 2.2.3, is a data structure that summarizes the data paths in one or several XML trees. The construction of the DataGuide is linear in the size of the call result. The DataGuide allows us to iterate through all the paths from the root to a text node. In Figure 3.7 is presented the DataGuide extracted from the example call result of Figure 3.2.

Generating KB Paths. Next, we generate the paths in the KB from x to literals. We compute the pairs of a path p' and a KB node y' (p', y'), *s.t.* $y' \in p'(x, y')$ and y' is a literal (Definition 2.1.6). As per Observation II, we limit the search to paths of length smaller or equal to three [81]. Hence, the results can be computed in a depth first traversal of the KB graph originating at x.

However, even short paths may lead to a huge number of results. For instance, $livesIn.hasNationality^-$ connects a person to all the other persons who have the nationality of the country where the person lives. Hence, the path connects “unrelated" entities via the very general concept `Country`. Our goal is to exclude relationships with an absurdly high number of outgoing edges such as $hasNationality^-$, $hasYearOfBirth^-$, $hasGender^-$. For this purpose, we discard paths with relations whose functionality is below a threshold α. In our experimental evaluation we present that by assigning a very small number in

this threshold (less than 10%) proved to be safe as no results were lost.

Path Pairs. The next step generates path pairs that are candidates for the alignment. Two paths are candidates if they select *at least* one value in common. Given the results of the previous steps for the sample, it computes pairs of form

$$(p, p') \; s.t. \; p(\lambda(x), y), p'(x, y') : y = y' \tag{3.1}$$

This means that we implicitly align the input entity x with the root $\lambda(x)$ (Observation II).

String normalization. Note that we need to check the equality of two strings ($y = y'$). However, caution is needed when comparing text values extracted from the XML result to literals in the KB, because the two sources may use different writings. For example, the XML result may contain the string "Sinatra, Frank", while the KB may contain the string "Frank Sinatra". The alignment of strings across such differences is a well-studied field of research. In our implementation, we considered a version of *bag-of-words* model. Two strings are considered as equal if they share the same words, independent of punctuation signs, word ordering, and word capitalization style. For this purpose, we normalise the values by lowercasing the string, reordering the words in alphabetical order, and omitting punctuation. Let *norm* be the normalisation function. If we apply the *norm* to the song title *"The World We Knew"* we will get:

norm("The World We Knew") = "knew the we world"

Hence, in Equation (3.1), the equality $y = y'$ is replaced by $norm(y) = norm(y')$.

Aligning Paths with Common Values. A simple pairwise comparison of text nodes in the call result to literals in the KB can be highly inefficient, since the number of pairs can be very high. Thanks to our normalisation, however, a pairwise comparison for similarity is no longer necessary: two strings are either identical or not considered similar.

For every sample entity, we construct two inverted indexes: one for the WS paths (extracted from the call result) and one for the KB path originating at the input entity.

For the index of the call result, a key is the normalized value of a text node, and the value is the list of paths from the root that select this key. For instance, for *"knew the we world"*, the entry in the index (p, y) is:

$$(\{/r/a/t/\tau\}, \text{"knew the we world"})$$

Similarly, the inverted index on the KB side will have as key the normalized value of a literal returned as a result, and as values the list of all KB paths that return this key. For instance, for *"knew the we world"*, the index (p', y') contains the following entry:

$$(\{created.label, hasChild.created.label\}, \text{"knew the we world"})$$

Note the same literal is selected by the KB path $created.created^{-}.created.label$. In fact, the results of this path subsume the results of the path *hasChild.created.label*. This is

because, unlike call results, KBs may contain cycles. The presence of cycles represents a subtle problem. One solution could be to eliminate all the paths with subpaths of form *r.r-*. However, this would result in missing interesting mappings such as *coauthor* = *created.created*$^{-}$. Our solution is to eliminate only the KB paths that include cycles in the level of data. To this purpose, inequality constrains are added to the body of the KB path so that we do not consider KB paths that visit the same entity twice.

Therefore, we can use a merge-join algorithm over the two indexes. We sort each of the two sets of pairs (p, y) and (p', y'), produced in the previous steps by the normalised value of the second component (y and y' respectively). Then we can merge the two lists in order to produce the results. The complexity is $\mathcal{O}(nlog(n))$ where n is the cumulated size of the two lists. This yields a set of pairs of the form (p, p'), where p is a path from the call result and p' is a path from the KB and both lead to the same value. These are our candidates for the path alignment.

3.3.4 Path Alignment

For a set of samples S, and a candidate pair of paths (p, p'), we need to compute a confidence score for the alignment of p and p' with respect to S. We present two strategies to this end.

Overlapping

Method. The simplest method to align a WS path p with a KB path p' is to use the overlapping of the results. We say that p' *overlaps with* p, if

$$\exists x \in S \; \exists y : p(\lambda(x), y) \wedge p'(x, y)$$

For simplicity, we use y to denote a literal and a text node with the same (normalised) value. An overlap can be stronger or weaker, depending on the proportion of samples that overlap. We compute the *confidence* of the overlap as the number of samples for which their results overlap, divided by the total number of samples:

$$conf(\,(p, p')\,) := \frac{\#x : \exists y : p(\lambda(x), y) \wedge p'(x, y)}{|S|}$$

Weaknesses of the Method. Although this simple method leads to remarkably good results, some alignments are lost. For instance, consider the alignment:

$$r/a/s/l/t/e/\tau \quad diedOnDate$$

Since we divide with the number of samples S, if the *living singers* represent more than half of the input entities selected for sampling, then this alignment is lost. Hence, we also experimented with a second strategy for path alignment, which is based on subsumption.

Subsumption

Method. The subsumption strategy aligns two paths if the results of one are included in the results of the other. We are interested in mining KB paths subsumed by WS paths and vice-versa, WS paths subsumed by KB paths. Let p and p' be two paths. We say that p' *subsumes* p, if

$$\forall x, y : p(x, y) \Rightarrow p'(x, y)$$

Again, x denotes the input entity and the root to which it is initially mapped in the call result, and y denotes both a literal and the equivalent text node. For instance, consider the following path pair:

$$/r/a/s/n/\tau \quad label$$

The WS path selects the name of the input entity for which albums are returned as well as the names of the other persons who released the respective albums. However, the KB path selects only the name of the input entity. Hence we have:

$$label \Rightarrow /r/a/s/n/\tau$$

Confidence. There are different ways to compute the confidence of such rules [78, 31, 47]. The problem in our case is that our data on both sides is incomplete: When the KB does not contain a certain fact, the WS can still return it, and vice versa. Hence, we opted to use the *PCA confidence*, a measure developed in [47]. This measure is particularly suited for incomplete relations. The measure assumes that a source knows either *all* or *none* of the p-attributes of some x. Under this assumption, the formula counts as counter-examples for a rule $p(x, y) \Rightarrow p'(x, y)$ only those instances x for which the query paths p return an answer, but where y is not among these answers. The number of counter-examples is computed, as following:

$$counter(p \Rightarrow p') := \#(x, y) : p(x, y) \wedge \exists y_2, y_2 \neq y : p'(x, y_2) \wedge \neg p'(x, y)$$

This yields the following confidence measure:

$$pcaconf(p \Rightarrow p') := \frac{\#(x, y) : p(x, y) \wedge p'(x, y)}{\#(x, y) : \exists y_2 : p(x, y) \wedge p'(x, y_2)} \tag{3.2}$$

The numerator is the number of instantiations shared in common by p and p':

$$overlap(p \wedge p') := \#(x, y) : p(x, y) \wedge p'(x, y)$$

However, the denominator represents the sum between the number of counter-examples and the overlap. Hence, $pcaconf$ can be also written as follows:

$$pcaconf(p \Rightarrow p') := \frac{overlap(p \wedge p')}{overlap(p \wedge p') + counter(p \Rightarrow p')} \tag{3.3}$$

This formula gives high confidence even if the alignments have small overlap. For instance, in our experiments, we found that the alignment

$$diedOnDate \Rightarrow r/a/s/l/e/\tau$$

achieves the highest confidence ($pcaconf = 1$). Hence, we can use *pcaconf* to spot possible alignments even if there is very small overlap. In such cases, the alignment is very risky. Hence, we have to validate such alignments through more probing. This works as follows: Whenever a path alignment achieves high PCA confidence, we select for probing, entities for which the newly discovered KB property is defined. Then we re-probe the WS with these entities.

The result of the path alignment is a set of pairs of the form (p, p'), where p is a path in the call result and p' is a path in the KB. Figure 3.8 shows the alignment for our running example.

3.3.5 View and Transformation Function

Branching Points. The path alignment has given us pairs of WS paths and KB paths that overlap in their query results. Now, we wish to align portions of the WS path to relations in the KB.

In the first step, we annotate the WS paths and the KB paths with branching points (symbolized by the symbol '$*$' in Figure 3.8). A branching point in a path means that there are several outgoing edges in the graph starting from that point of the path that are labeled with the same name.

For example, following the path $r/a/t/\tau$ in the call result of Figure 3.2, we encounter several edges labeled with a. Hence, a is a branching point.

Formally, we say that l_i is a branching point for the WS path $/l_1/l_2 \ldots /l_{i-1}/l_i \ldots /l_n$ *iff*

$$\exists x \in S, z, y_1, y_2 \; : \; /l_1 \ldots /l_{i-1}(\lambda(x), z) \wedge \\ \wedge \, l_i(z, y_1) \wedge l_i(z, y_2) \wedge y_1 \neq y_2$$

We use a path summary in the form of a DataGuide tree, as it is described in chapter 2.2.3, to keep track of the nodes that occur under the same path and that have siblings labeled with the same name. The DataGuide is constructed for the call results of all the samples. Its construction requires only a depth first traversal of each sample tree and its size is small because it summarises only paths that consist of structure nodes. Then, we annotate with '$*$' the nodes of the DataGuide that are embeddings for two or several sibling nodes.

For the KB paths, we insert branching points whenever a relation is not a function. Some KBs explicitly declare their functions. If that is not the case, we compute the functionality for each relation (Equation 2.2). Since KBs may contain some noise, we insert branching points at a relation only if its functionality is lower than a threshold β.

Cut and Align. Once the KB path and the WS path are annotated with branching points, we want to align the path segments of the WS path with the segments of the KB path. Our input are the path alignments produced by the previous step, annotated with branching

/r/a/t/τ*	*created*.label*
/r/a/d/τ*	*created*.date*
/r/a/s*/l/b/τ*	*birthdate*
/r/a/s*/l/e/τ*	*diedOnDate*
/r/a/s*/n/τ*	*label*
/r/a/s*/g/τ*	*gender.label*

Figure 3.8: Results of the path alignment step for *getAlbumsByArtist*

(*r/a*, created*)	z
(*r/a*/s*,*)	x
(*r/a*/t/τ, created*label*)	y_1
(*r/a*/d/τ, created*date*)	y_2
(*r/a*/s*/l/b/τ, birthdate*)	y_3
(*r/a*/s*/n/τ, label*)	y_4
(*r/a*/s*/g/τ, gender.label*)	y_5
(*r/a*/s*/l/e/τ, diedOnDate*)	y_6

Figure 3.9: New path alignments and their associated variables for *getAlbumsByArtist*.

points, and our output will be more path alignments. The clue of our algorithm is the insight that if we have a path of functional relations on the KB side, then this path has to be aligned to a (sub-)path with no branching points on the XML side. This means that the rightmost '$*$' annotation of one path is aligned with the right most '$*$' of the other path in the pair. Let us write $P * p$ to denote a path such that $P*$ ends in a branching point or it is the empty path, and p has no branching points. Then we generate new path alignments by recursively applying the following rule:

$$(P * p, P' * p') \rightsquigarrow (P, P')$$

For the path alignments in Figure 3.8, this gives rise to the new alignments in the Figure 3.9.

Weakness of the Method. The annotation with branching points on the WS paths is biased by the set of WS call result samples. For example, if all call results return only singers that have released a single album, then our method will not annotate */s* as a branching point.

Duplicate Elimination. The previous step may result in associating the same KB path to different WS paths. This means that the same relationship appears multiple times in the call result. If this happens, we select the alignment that was generated from the pair with the highest confidence. Also, it may happen that several KB paths map to the same WS path. If the KB paths select literals, we use the same approach. If not, then they concern relationships between complex entities. For instance, we may have as candidates *created*

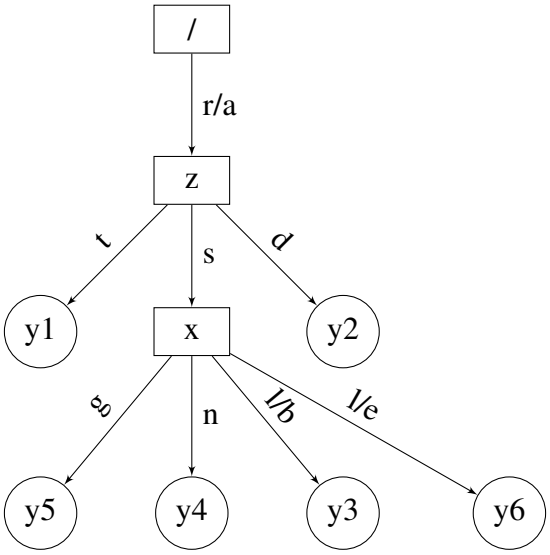


Figure 3.10: The tree-like hierarchy of paths in XML of Figure 3.9.

and *bornIn.producedIn*$^-$. In that case, we select the path where the maximal functionality of its relations is minimal.

View Definition. We now want to deduce the *view definition* from our path alignments. We introduce a new variable for each path alignment (shown in Figure 3.9). Now, the view definition is simply the join of the KB paths with the appropriate variables. More precisely, let us write $P' \rightarrow v$, if some alignment (P, P') is associated to a variable v. Then, we construct the body of the view definition with the following rules:

$$\left.\begin{array}{l} P* \rightarrow v_1 \\ P*p \rightarrow v_2 \end{array}\right\} \rightsquigarrow p(v_1, v_2)$$

$$\left.\begin{array}{l} P* \rightarrow v_1 \\ P*P'* \rightarrow v_2 \\ \nexists P''* \rightarrow v, P''' : P*P''*P''' = P*P'* \end{array}\right\} \rightsquigarrow P'(v_1, v_2)$$

The resulting atoms are added to the body of the view definition. The head atom of the view definition is the name of the WS. It contains only the variables that correspond to literals. This is because the entity identifiers used by the KB cannot be obtained by calling the WS. For our running example, we obtain exactly the view definition of Figure 3.4.

Transformation Function. We have shown how to derive automatically the view definition of a WS in the global schema. Now, we turn to the question of how we can transform a concrete call result into tuples of this view definition. Our idea is to use an XSLT script for this purpose. Given a call result of a WS, the script will extract bindings

for the variables in the head of the view definition of the WS. We now explain how to derive this script automatically from our path alignments.

We first organise the WS paths of the path alignment as a tree. The root of the tree is labeled with "/", and the other nodes represent variables. Edges are labeled with path queries. In the example of Figure 3.9, this leads to the tree shown in Figure 3.10.

Algorithm 1 transforms such a tree into an XSLT script. Figure 3.11 shows the result of the algorithm on the example tree. It requires as input the root node of the tree. We assume that the functions *isRoot(), isLeaf(), getChildren()*, and *pathFromParent()* are available for every node. Let *pathFromParent()* return the path query on the edge between the node and its parent. For leaf nodes, the function omits the last part "text()" (τ). For instance, y_1.*pathFromParent()=t*. Our algorithm performs a depth-first search on the tree, and outputs a variable assignment for each edge of the tree.

Algorithm 1: makeXSLT(TreeNode v)

```
   if v.isRoot() then
     print <xsl:template match="/">
     for v' : v.getChildren() do
       makeXSLT(v')
     end for
     print </xsl:template>
   end if
   if v.isLeaf() then
1    print {v:<xsl:for-each select="v.pathFromParent()">
     {<xsl:value select="." >} </xsl:for-each>}
   else
     print {<xsl:for-each select="v.pathFromParent()">
     for v' : v.getChildren() do
       makeXSLT(v')
     end for
     print </xsl:for-each>}
   end if
```

The output of this algorithm is an XSLT script, like the one depicted in Figure 3.11. This script is generated only once per WS. Each time a client calls the WS, the client has to execute the XSLT script on the call results, which will yield tuples in the view definition of the WS. We remark that this gives us an end-to-end solution for integrating Web services into a global schema: Given a schema and a WS, we can automatically derive a view definition of the WS, and a method to map the call results into that view definition.

3.4 Baseline Approach

We have presented an end-to-end solution for the mapping of a Web service into a global schema. We will now present a baseline solution to this problem, which is based

```
<xsl:template match="/">
 <xsl:for-each select="r/a"> {
  {y1: <xsl:for-each select="t">
     {<xsl:value-of select='.'/>}} </xsl:for-each>
  {y2: <xsl:for-each select="d">
     {<xsl:value-of select='.'/>}} </xsl:for-each>
  <xsl:for-each select="s" >{
    {y4: <xsl:for-each select="n">
        {<xsl:value-of select='./ >}} </xsl:for-each>
    {y3: <xsl:for-each select="l/b">
        {<xsl:value-of select='./ >}} </xsl:for-each>
    {y5: <xsl:for-each select="g">
        {<xsl:value-of select='./ >}} </xsl:for-each>
 </xsl:for-each>} </xsl:for-each>
</xsl:template>
```

Figure 3.11: XSLT Code for tree-like hierarchy of Figure 3.10.

on wrapper induction. We adopt the approach that [81] has developed for mapping Deep Web forms into a KB. The approach creates a wrapper that extracts a pseudo-KB on the result pages of the Deep Web form. This pseudo-KB is then aligned with the reference KB using the PARIS ontology alignment algorithm [102]. A similar approach has been presented in [29] for mapping XML documents to RDF ontologies.

In the spirit of wrappers, we construct entities for repetitive structures. In our case, these are subtrees rooted at nodes that can have sibling nodes labeled with the same name. In our running example (Figure 3.2), these are the nodes labeled with *a* and *s*. We use an XSLT script to implement this wrapper. Conveniently, we can use Algorithm 1 to generate this script. As input, we use the DataGuide that we computed for our samples. The branching points and the leaf nodes are annotated with variables. Two XML paths with the same labels are mapped to the same relation in the pseudo-KB. Running this script on the call results will yield a pseudo-KB, much like in [81]. Then, we use PARIS to align this pseudo-KB with the reference KB.

Although this solution may seem reasonable, our experimental evaluation shows that it does not lead to good results. One problem with this approach is that two entities in the reference KB may map to the same entity node in the pseudo-KB. Another problem is that a single relation in the pseudo-KB may correspond to an entire path of relations in the reference KB. This derails KB alignment approaches such as PARIS, which assume that a single relation in one KB corresponds to a single relation in the other KB.

3.5 Prototype

In this section, we provide a description of the interface of our DORIS system. Using the graphical interface, the user will be able to trace every step that DORIS takes. The

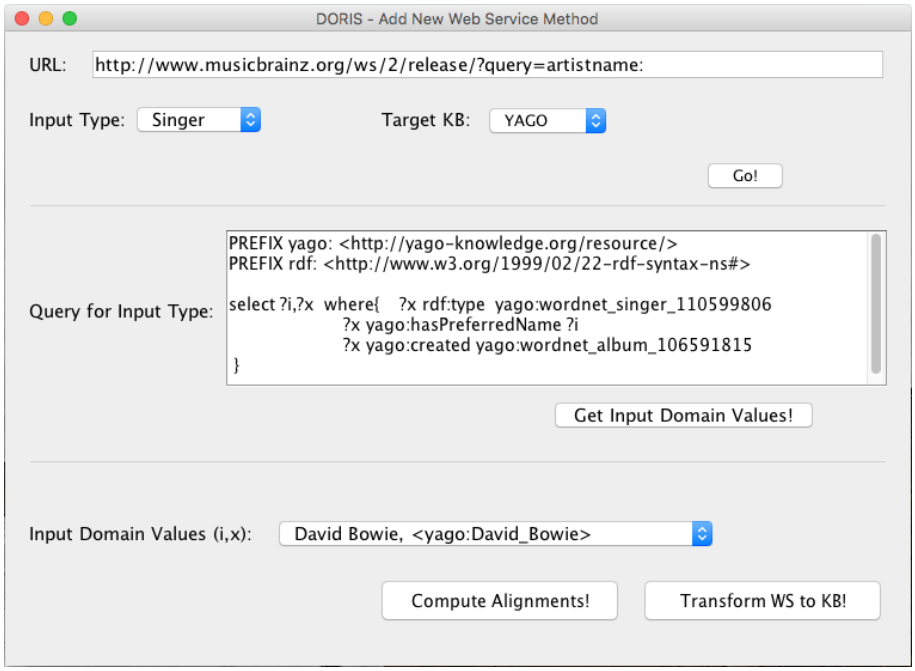


Figure 3.12: DORIS: Main Interface.

interface illustrates a suite of strategies that we have investigated, and the design decisions we made.

The very first interface that the user interacts with is shown in the Figure 3.12. First, the user selects a target KB from the pop up menu. Our demo currently supports DBpedia, YAGO, and the KB of the French National Library, BNF. Next, the user chooses a WS that she wishes to map to a KB. This can be any WS that the user came across albeit the KB must stores facts for the topic of the WS. As shown in Figure 3.12, DORIS requires as input the URL of the WS and the type of the input. For instance, let us also consider the service with URL:

$$http://www.musicbrainz.org/ws/2/release/?query = artistname$$

and let the type of the input be the class `Singer` from the target KB. After the selection of the input type, and the target KB, in the text area of the interface appears the query that is addressed to the target KB in order to get the input type values (i, x) which are used to probe the Web service. These input values are presented in the pop up menu and they are available for the user to revise.

Compute Alignments! and *Transform WS to KB!*. The first one allows the user to investigate the path alignment step that aligns classes and relations. The second option will use the alignment of classes and relations and will illustrate the computed parameterized query and XSLT script that transforms the call result in terms of the KB.

In more details, the interface that demonstrates the computation of alignments is presented in Figure 3.13. Here, the user may select the threshold she wants to apply for the overlapping and the subsumption methods. After the selection of the thresholds from the

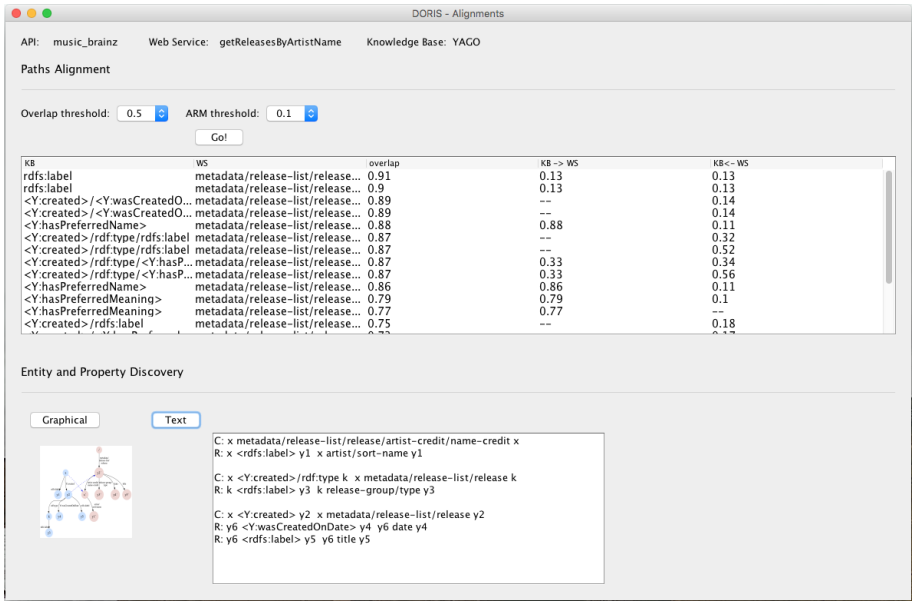


Figure 3.13: DORIS: Alignment Interface.

user, the system allows her to investigate the results of the algorithm for path alignment. In the first text area appear the aligned paths between the knowledge base and the Web service along with the scores for the two methods. *Overlap* for the overlapping method and $KB \rightarrow WS$, $WS \rightarrow KB$ correspond to the both ways subsumption method.

Apart from the paths alignment, in the same interface (Figure 3.13) is provided to the user the Class and Atom alignment result in two different ways. User may select between graphical representation and textual representation. By clicking on *Graphical*, the graph is maximized and the user may observe in details how the two sources are aligned. The two sources are depicted with different colors, in cycles are presented the variables assigned by the algorithm after the *Cut and Align* step. Blue arrows connect the nodes that represent entities of the aligned classes. If the user selects *Text*, will get the same result in textual format this time.

After the computation of the alignments, the user can pass to the next interface presented in Figure 3.14. Here, the user is observing the view (upper part of the interface) and the transformation function in the form of an XSLT script (lower part of the interface). Both are computed using the alignments of the previous step.

3.6 Experimental Evaluation

Our algorithms are fully implemented in a system called *DORIS* (Discovering Ontological Relations In Services), presented above.

Data Sources. We have tested DORIS on more than 50 real-world WSs provided by more than 10 APIs. The data exported by the WSs cover four domains: music, movies, books, and geographic data. Table 3.2 shows the WS APIs that we considered, grouped

Figure 3.14: DORIS: Transformation Interface.

by thematic domain. The first column shows the APIs. The third column shows the Web services offered by these APIs. Several APIs can provide the same Web service. For example, both book APIs support *getAuthorInfoByName*. Therefore, the second column shows how many APIs support a given WS.

We have conducted experiments for three target KBs: YAGO [103] and DBpedia [13] (which are extracted from Wikipedia), and BNF (bnf.fr) (a KB from the domain of books curated by the National Library of France). We probed each Web service for 100 samples.

Platform. Our system is implemented in Java (JVM 1.7). We use standard Java libraries to parse the XML and JSON call results. We use the Jena 2.11 library to store the KBs, and the Jena SPARQL functionality to query them. We ran all experiments on a personal laptop (2,9 GHz Intel Core i7, 8GB). Our method takes roughly 2 minutes to derive the schema of a WS (Path Alignment and Class & Atom Alignment).

Path Alignment

Our first series of experiments concerns the path alignment (Section 3.3.1). We probed each WS with instances of the input domain from the YAGO knowledge base. As a gold standard, we designed the correct path alignments manually for each WS of each API.

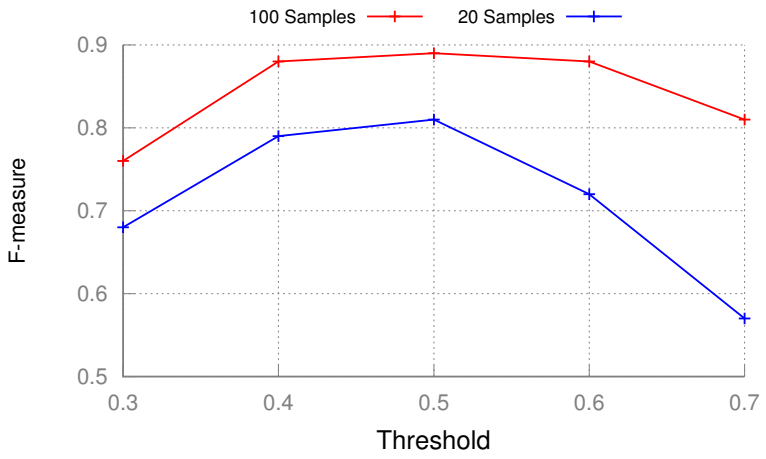


Figure 3.15: Average performance of Path Alignment.

Our first goal is to determine the right threshold for the overlapping algorithm. We examine the performance of our method with respect to different threshold values. Figure 3.15 shows the average F-measure across all WSs for the alignment under different thresholds. The threshold 0.5 produces the best results when was tested on 20 samples and 100 samples, hence we use it for what follows.

Then, we ran experiments for both alignment strategies: the overlapping and the subsumption strategy. For the subsumption, we checked whether the results of the KB path are subsumed by the results of the WS path ($KB \rightarrow WS$), and vice-versa ($WS \rightarrow KB$). Table 3.2 shows the precision and the recall for each WS (averaged over the APIs that provide the WS).

Our experiments show that the overlapping strategy (Overlap columns) can align paths with a precision (P) of 57%-100%. In the vast majority of cases, the precision is over 80%. Recall (R), likewise, is well over 90% in the majority of cases.

The subsumption strategy performs slightly worse. The main reason is that the PCA confidence does not penalise alignments that have the support of a small number of samples. This yields many spurious matches. On the other hand, this allows aligning paths with incomplete relations that are otherwise lost. For instance, we have correct alignments for the following relations: *diedOnDate*, *wasCreatedOnDate*, *hasLongitude*, and *hasLatitude*. Table 3.1 shows the alignments that only rule mining can find for all the API we have tested. In order to find these alignments also with the overlapping method, more samples would be needed.

Class and Atom Alignment

We refer to the bindings of a variable in the view using the term class. By atom we mean an atom of the view. We say that a class or an atom are correctly aligned if the transformation function extracts correct values for them. In order to evaluate the class and atom alignment, we first generated the ideal class and atom alignment manually. Then, we ran our alignment algorithm and compared the results to this gold standard. As input,

Relation	Method	API
diedOnDate	getArtistInfoByName	music_brainz
diedOnDate	getArtistInfoById	
diedOnDate	getReleasesByArtistId	
wasCreatedOnDate	getReleasesByName	
hasLongitude	getCityByName	geonames
hasLatitude	getCityByName	
diedOnDate	getActorInfoById	themoviedb

Table 3.1: Alignments discovered only by the subsumption strategy of Section 3.3.4.

we used the path alignments computed by the overlapping strategy.

Duplicate Elimination. We first wanted to see whether duplicate elimination helps precision and recall for the class and atom alignment (see Section 3.3.5). Hence, we ran the alignment with and without duplicate elimination and measured precision and recall. Our experiments show that the technique helps: By removing the duplicates, the average precision across all Web services for YAGO relations rises from 45% to 91%. At the same time, recall is almost the same. It decreases from 95% for the algorithm that allows duplicates to 93% for the one that removes them. The precision and recall for classes remains unchanged. Therefore, we decided to remove duplicates in what follows.

API		No	WS	Path Alignment: **Overlapping** P	**Overlapping** R	KB→WS P	KB→WS R	WS→KB P	WS→KB R
GeoData	geonames	1	getCityByName (*City*)	**1**	**0.73**	0.17	0.33	0.0	0.0
		1	getCountryByName (*Country*)	**0.62**	**0.71**	0.25	0.2	0.0	0.0
Books	isbndb	2	getAuthorInfoById (*Id*)	**1**	**1**	0.29	0.49	0.35	0.89
	library_thing	2	getAuthorInfoByName (*Author*)	**0.97**	**0.97**	0.32	0.48	0.23	0.5
		2	getBookInfoByTitle (*Book*)	0.84	**1**	0.85	0.72	0.5	0.5
Movies	themoviedb	1	getActorInfoById (*Id*)	**1**	**0.8**	0.33	0.8	0.23	1
		1	getActorInfoByName (*Actor*)	**1**	**1**	0.33	0.67	0.25	1
		1	getMoviesByActorId (*Id*)	0.88	**1**	0.27	0.43	1	0.57
Music	music_brainz	6	getArtistInfoById (*Id*)	**0.82**	**0.89**	0.32	0.49	0.33	0.81
		1	getArtistInfoByMBID (*Id*)	**0.57**	**1**	0.2	0.5	0.19	0.75
		6	getArtistInfoByName (*Singer*)	**0.9**	**0.97**	0.33	0.49	0.46	0.31
	musixmatch	4	getAlbumByTitle (*Album*)	0.84	**0.75**	0.87	0.17	0.2	0.12
	last_fm	4	getAlbumsByArtistId (*Id*)	**0.94**	**0.99**	0.22	0.2	0.46	0.57
	echonest	1	getAlbumsByArtistMBID (*Id*)	**1**	**1**	0.33	0.2	0.5	0.6
	deezer	2	getAlbumsByArtist (*Singer*)	**0.85**	**0.91**	0.29	0.31	0.67	0.38
	discogs	1	getTopTracksByArtistId (*Id*)	**1**	**0.93**	0.33	0.13	0.6	0.6
		4	getTrackByTitle (*Song*)	0.81	**1**	0.94	0.53	0.25	0.12
		2	getTracksByArtistId (*Id*)	**0.88**	**1**	0.17	0.22	0.17	0.33
		4	getTracksByArtist (*Singer*)	**0.91**	**0.94**	0.29	0.29	0.52	0.41

Table 3.2: Path Alignment on the YAGO KB.

API	No	WS	Classes DORIS P	Classes DORIS R	Atoms DORIS P	Atoms DORIS R	Atoms *PARIS* P	Atoms *PARIS* R
GeoData { geonames	1	getCityByName ($City$)	**1**	**1**	**1**	**0.67**	0.31	0.67
	1	getCountryByName ($Country$)	**1**	**1**	**1**	**1**	0.1	0.33
Books { isbndb	2	getAuthorInfoById (Id)	**1**	**1**	**1**	**1**	0.33	0.37
Books { library_thing	2	getAuthorInfoByName ($Author$)	**1**	**1**	**1**	**1**	0.41	0.62
	2	getBookInfoByTitle ($Book$)	**1**	**1**	**1**	**1**	0.29	0.5
Movies { themoviedb	1	getActorInfoById (Id)	**1**	**1**	**1**	**0.67**	0.4	0.67
	1	getActorInfoByName ($Actor$)	**1**	**1**	**1**	**1**	0.33	1
	1	getMoviesByActorId (Id)	**1**	**1**	**1**	**1**	0.25	0.67
Music { music_brainz	6	getArtistInfoById (Id)	**0.89**	**0.78**	**0.83**	0.73	0.38	0.72
Music { musixmatch	1	getArtistInfoByMBID (Id)	**1**	**1**	**1**	**1**	0.33	1
Music { last_fm	6	getArtistInfoByName ($Singer$)	**0.89**	**0.94**	**0.84**	0.9	0.28	0.72
Music { echonest	4	getAlbumByTitle ($Album$)	**1**	**1**	**1**	**1**	0.09	0.15
Music { deezer	4	getAlbumsByArtistId (Id)	**1**	**1**	**1**	**1**	0.29	0.37
Music { discogs	1	getAlbumsByArtistMBID (Id)	**1**	**1**	**1**	**1**	0.5	0.5
	2	getAlbumsByArtist ($Singer$)	**0.88**	**1**	**0.88**	**1**	0.24	0.33
	1	getTopTracksByArtistId (Id)	**1**	**1**	**1**	**1**	0.33	0.25
	4	getTrackByTitle ($Song$)	**0.79**	**1**	**0.86**	**1**	0.3	0.32
	2	getTracksByArtistId (Id)	**0.88**	**1**	**0.88**	**1**	0.25	0.66
	4	getTracksByArtist ($Singer$)	**0.83**	**1**	**0.81**	0.95	0.23	0.32

Table 3.3: Class & Atom Alignment on the YAGO KB.

YAGO. The results of aligning the Web services to the YAGO knowledge base are presented in Table 3.3. We report that our system achieves a perfect precision and a perfect recall on nearly all WSs. Only very few WSs could not be mapped entirely correctly. We reckon that this is the first time that the result of WSs can be mapped fully automatically to the schema of a target KB.

Other KBs. We have also tested our algorithm on DBpedia and BNF. For DBpedia, we used all 50 WSs. For BNF, which is a domain-specific KB, we ran only the WSs from the Books domain. We present an average of the precision and recall of our class and atom alignment in Table 3.4. Since the BNF contains data that overlaps a lot with the WS, we obtain a perfect precision and recall for this KB. The other alignments are also of very respectable quality, with recall and precision values well over 90%.

	Precision		Recall	
	Classes	Atoms	Classes	Atoms
YAGO	0.92	0.91	0.96	0.93
DBpedia	0.91	0.92	0.98	0.95
BNF	1	1	1	1

Table 3.4: Average Performance of Alignments

Comparison to Baseline. In Section 3.4, we introduced a baseline approach to the problem of WS alignment. This approach transforms the WS call results into a pseudo-KB, and uses a state-of-the-art alignment approach (PARIS [102]) to align the pseudo-KB to the reference KB.

We ran PARIS with exactly the same data as DORIS. PARIS computes a confidence score for each alignment. We used a threshold of 0.6 on this score to determine the final output, which was the value for which PARIS performed best. The results of this approach are shown in the last two columns of Table 3.3. As we see, DORIS outperforms PARIS by a huge margin. DORIS achieves near-perfect alignment. In the case of PARIS, the precision and recall scores are in the range between 30%-60%. The poor performance of PARIS is explained by the fact that it in order to perform well, the schemas of the two ontologies need to be similar. However, this is not the case at all in our problem. Moreover, PARIS cannot discover complex relations alignments like *hasGender.label*, while DORIS discovers them.

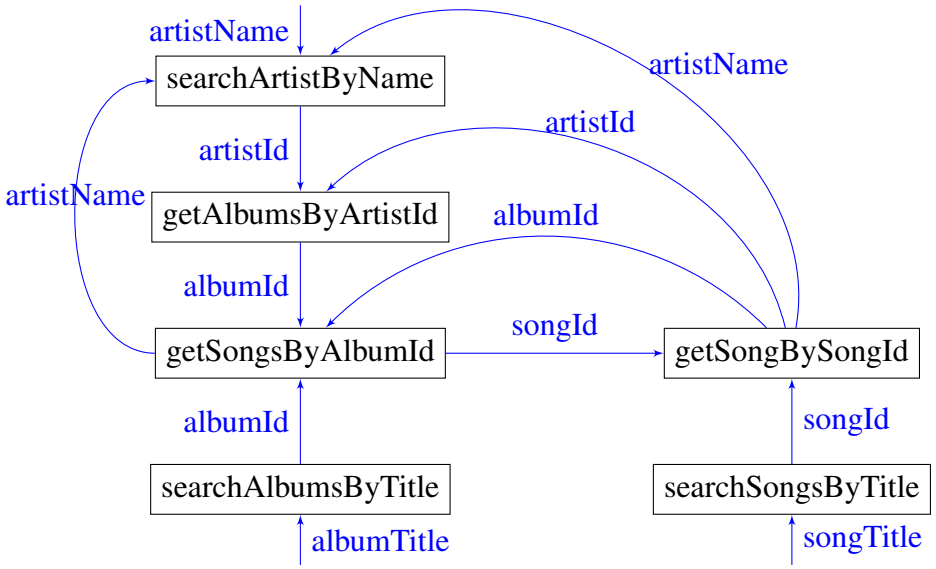


Figure 3.16: I/O Dependencies between the Web services of *musicbrainz* API.

3.7 Discovering I/O Dependencies

Apart from the global schema, the input values one needs to call the Web services can be found in other sources. These sources can be the Web in general, or even the same API that provides the Web service. In this case the input values can be encoded in the call result of another Web service. For example, one Web service delivers singers, another Web service could use this output in order to find the albums of these singers. We call this dependency between services *I/O dependency*.

A specific category of I/O dependencies is the one concerning identifiers (IDs). Many

API providers use internal IDs rather than names to identify entities. IDs avoid ambiguity and provide users with an easy navigation mechanism. Since IDs are internal to the API, they can almost never be obtained from external sources (such as the KB). However, if they are contained in the result of some Web service of the same API, they can be inferred by joining the data of the two Web services.

Figure 3.16 shows the I/O dependencies of some of the services in the *MusicBrainz* API. Each node represents a WS where the incoming edges to a WS are labeled with its input type. Some of the types are unknown to the KB. Quite possibly, some I/O dependencies were deliberately designed to unveil the data only gradually, so that an exhaustive mirroring of the data is prevented. In our example, three calls are necessary to retrieve the songs of a given singer. First, we have to call *searchArtistByName* to obtain the ID of the singer. Then, we have to use this ID to call *getAlbumsByArtistId*. This gives us the IDs of the albums by this singer. Finally, we can call *getSongsByAlbumId* with the each of the album IDs, so that we can obtain the songs on each album.

While some Web services may be joinable, other may be incompatible. For example, it does not make sense to use the output of a Web service that delivers albums id as inputs to a Web service that expects movies id. We will now show that our approach for mapping Web services into a global schema can also be used to determine the I/O dependences between Web services of the same API.

3.7.1 Problem Statement

Let ρ_f be a function that maps a variable in the view of a WS f to its bindings in our samples of f. We consider the following problem: Given a WS f_O that has been mapped to the KB, and given another WS f_I with only one input of unknown type, compute I/O dependencies of the form:

$$(f_O, x, p, f_I)$$

Here, x is variable in the view of f_O. p is an WS path such that $\forall x' \in \rho_{f_O}(x)$, $p(x')$ has values for the input of f_I and f_I returns data relevant to x'.

Example 1. Let f_O=*getAlbumsByArtist* and let f_I be a WS that returns songs by album-id. We aim to discover the following I/O dependency: $(f_O, z, id/\tau, f_I)$. Here, z is the variable used for album entities in the view in Figure 3.4. In the call result depicted in Figure 3.2, z had as bindings the two nodes labeled with a. Note that the path id/τ selects exactly the ids of the respective albums.

3.7.2 Approach

Mini-KB Extraction. The interesting aspect of our example I/O dependency is that it concerns paths in f_O that were not yet mapped to KB relations (namely id/τ). Our goal is now to extend our view definition and our transformation function to these unmapped paths. For each variable x in the view of f_O, and for each WS path p' s.t. $\exists x' \in \rho_{f_O}(x) \wedge p'(x') \neq \emptyset$, we add a new relation name to the schema of the KB, and we map p' to that new relation name. The XSLT script for f_O is updated accordingly. Let *mini-KB(x)* be a

KB computed by applying the updated XSLT script to every call result in our sample for f_O.

Generating candidates. A candidate I/O dependency of the form (f_O, x, p, f_I) is computed for every variable x from the view of f_O and every path p in f_O where

$$(\#x' : x' \in \rho_{f_O}(x) \wedge p(x') \neq \emptyset) > \theta$$

In our experiments, we set $\theta = 20$. For probing f_I, we extract from *mini-KB(x)* tuples of the form:

$$(x^k, l) : \exists x' \in \rho_{f_O}(x) \wedge \sigma'_{f_O}(x') = x^k \wedge p(x', l)$$

where l serves to call f_I, but we see x^k as the input entity of that respective call. σ'_{f_O} is a function that gives the entity that corresponds to a node x' from a call result of f_O. We now discuss when we keep a candidate I/O dependency (f_O, x, p, f_I).

Step 1: Catch Error Messages. If f_I returns systematically an error message, we discard the candidate. However, some WSs will not return an error message if fed with a bad input entity. Instead they will return results for all entities whose names or other attributes are similar to the input entity. Therefore, our first step cannot filter out all bad candidate I/O dependencies.

Step 2: Checking Input's Position in the Output. If the call is valid, then it should return information about the input value. This value should consistently appear under the same path(s) from the root (Observation IV). Assume that we discover that the input value occurs *always* under the path p_i in the call results of f_I. Then, we can filter out every candidate (f_O, x, p, f_I) that did not discover p_i as being the path with the inputs.

Step 3: Align f_I and the mini-KB. Finally, we use the path alignment algorithm from Section 3.3.1 to align label paths in the outputs of f_I with path relations in *mini-KB*(x). We rank the candidates between f_I and f_O according to the number of paths produced by the alignment. The intuition is that f_I should return some of the properties of x returned also by f_O. We select as solution the top candidate, meaning, the path p that leads to the highest overlap.

Example 9. *Consider again our Web service f_O = getAlbumsByArtist. Let z be the class of albums (denoted also with z in our example) and p=id/τ be the path query selecting the ids of the respective albums. Assume now that the selected values are injected as inputs to f_I = getSongsByAlbumId. If, among others, the new function returns also the title, the release date, and the singer of the album, then the path alignment algorithm would produce the alignments shown in Figure 3.17.*

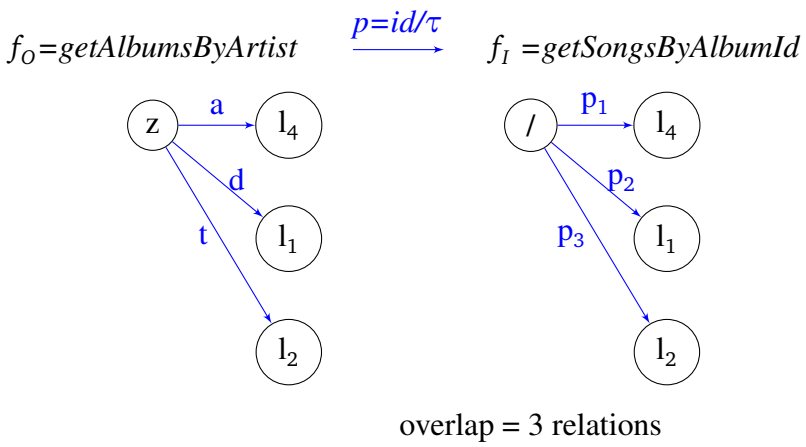


Figure 3.17: Aligning f_I and f_O.

Discovering Types for WS Inputs. Let our candidate (f_O, x, p, f_I) be a I/O dependency. Hence, f_I returns data relevant to x. Thus, we can say that the input of f_I has as type the class of x^k (or x) in the KB. However, the relation that links x^k to l in the KB might not be the default relation *label*. Hence, the input of f_I is annotated with the class of x^k (its input entity) and the (new) relation to which p is mapped to. With this, f_I is now ready to be mapped to the target KB.

3.7.3 Experimental Evaluation

Our algorithm for discovering I/O dependencies is implemented in the same platform used for DORIS system and described in Section 3.6. The 50 real Web services that were used for the evaluation of this algorithm are provided by more than 10 APIs and cover four domains: music, movies, books, and geographic data.

Discovering I/O Dependencies Here we evaluate our algorithm for discovering I/O dependencies for WSs whose input type is hidden. For each API, we determined the dependencies manually. Then, we ran our algorithm for each API, and compared the results to the true ones. Table 3.5 shows the precision and the recall for several variants of the algorithm: S1 and S2 implement Step 1 and Step 2 respectively. S23 implements Steps 2 and 3 together. We find that Step 1 sometimes excludes good answers, which leads to a drop in recall. The good news is that all strategies have a very high recall, which is almost perfect for S2 and S23. Furthermore, S23 achieves a precision that is consistently over 80%. Hence, our method is able to discover I/O dependencies and input types fully automatically with high accuracy.

3.8 Summary

In this chapter, we have shown how to map the results of a Web service fully automatically into the schema of a knowledge base. Our approach aligns the call results of

API	Precision			Recall		
	S1	S2	S23	S1	S2	S23
deezer	0.21	0.67	0.95	0.77	1	1
discogs	0.27	0.83	0.81	1	1	1
echonest	0.78	1	1	1	1	1
isbndb	0.48	0.82	0.82	1	1	1
last_fm	0.24	1	1	1	1	1
library_thing	0.57	0.6	1	1	1	1
music_brainz	0.1	0.84	0.84	0.93	0.93	0.93
musixmatch	0.11	0.55	0.83	1	1	1
themoviedb	0.19	0.78	0.89	1	1	1

Table 3.5: I/O Dependencies Discovery

the Web service to relations of the KB, it derives a view definition of the service in the schema of the KB, and it produces an XSLT script that transforms the call result in the terms of the KB. Our experiments show that our approach produces near-perfect results on over 50 real Web services in a variety of domains.

Moreover, we have shown how to discover I/O dependencies between different Web services of the same API by using our approach on mapping services to a global KB schema. Especially in REST services where the absence of description for the input and output parameters of a service is habitual, discovering the type of the input that a specific Web service expects is really crucial. With this work we take a step further from making a single service inter-operable and we give the potential to the users to explore the maximum that an API can give by accessing all its Web services.

We hope that our techniques can provide a bridge between different data providers, and thus help the rise of tomorrow's knowledge-centric applications.

Chapter 4

SOFYA: Online Relation Alignment for Linked Datasets

4.1 Introduction

Since the invention of Linked Data, the number of datasets exposed as linked data have been growing exponentially. The analysis in [1], estimates that currently there are more than 1000 datasets, with roughly 30 billion facts in the form of triples.

These datasets, span across multiple domains. Based on [1], the majority of datasets fall into the *Governmental* and *Social Web* domains, with 18% and 51%, respectively, followed by *Publications*, *Cross-domain* and *Life sciences*. Most of those datasets have different *schemas* and *vocabularies* to represent the information they embed. From the thousands of datasets, in LOD there are approximately 650 vocabularies used to represent their data [1]. However, based on [97], only a small percentage of roughly 2% of schemas are aligned across the different datasets. Finally, these datasets usually are exposed through SPARQL endpoints.

The diversity and richness in information available in LOD has led to some very successful examples. Some of the most prominent examples like the *Google Knowledge Graph* [39] used to provide factual information for *entity search*. A similar initiative is exploited by the project *EntityCube-Renlifang* at Microsoft Research [80], or a comparable initiative at Yahoo! [21]. The common intuition and motivation across all such projects is the combination of multiple LOD, like IMBD [1], DBpedia [13], Freebase [2], etc., into a coherent knowledge graph which serves as a backbone for Web search.

However, integrating and combining multiple datasets from the LOD is challenging. One of the main challenges for such integration is the diversity in terms of schemas. This has led to a very active research initiative, namely *Ontology matching* with the aim of mapping classes/concepts defined across disparate schemas. A recent survey [99] shows a large number of approaches that mostly deal with the alignment of classes/concepts from schemas. Yet, the problem of aligning *relations* (i.e. predicates) has not seen much progress. Furthermore, disregarding the distinction between class and relation alignment

1. `http://www.imdb.com`
2. `http://www.freebase.com`

for linked datasets, one crucial challenge that is not addressed by previous works, is the applicability of such approaches when there is limited access to data sources. Existing approaches require full-access to the datasets, and the algorithms are performed on a dataset snapshot.

In this Chapter, we address the problem of *relation alignment* for linked datasets. We assume an *online* setting, where the access to a remote and *knowledge base* (e.g. DBpedia, Freebase, or YAGO) is only possible through a SPARQL endpoint. Hence, knowledge bases can only be queried through SPARQL expressions. Given a relation from a *source* knowledge base, e.g. `wasBornIn` (describing the birth location of a person), our goal is to find *target* relations in which `wasBornIn` is subsumed in a target knowledge base.

We propose a supervised machine learning approach for relation alignment. We learn the models by computing features on data samples which we extract from the KBs through their SPARQL endpoints. The features fall into three main categories, namely, measure the overlap in terms of triples between two relations, the relation name similarity and other general statistics such as the type distribution from the subjects or objects of the respective relations.

One main advantages of the proposed approach in *SOFYA*, is that one does not need to download and store locally the knowledge bases. All the computations are made on data that is extracted though querying the datasets. Furthermore, our approach can be used even if the data transfers from the respective knowledge bases are further limited. We propose a sampling strategy that carefully selects a small data sample from the two knowledge bases which guide the alignment process. Hence, our method accounts for the efficiency of the relation alignment process. Furthermore, by performing the alignment online, we compute the alignments on the fly and thus account for the evolving nature of the datasets.

The remaining of this chapter is organised as follows. Section 4.2 formally describes the problem that we tackle in this Chapter. Section 4.3 details our approach on how to discover relation alignment candidates for a relation from a source knowledge base in which the relation is subsumed in a target knowledge base. In Section 4.4 we show the efficiency precautions we undertake in our approach. In Section 4.5 we present the experimental setup for the relation alignment approach and in Section 4.6 we carry out an extensive experimental evaluation, where we assess the efficiency and effectiveness of our approach, before we conclude in Section 4.7.

4.2 Problem Statement

In this section, we define the problem setup for the *relation alignment*. In the remaining of this chapter, we will consider a pair of knowledge bases (KB), a source $KB_S(R_S, P_S, L_T)$ and a target $KB_T(R_T, P_T, L_T)$, which can be queried through their respective SPARQL endpoints. Specifically, each KB is represented by a set of triples $\langle subject, predicate, object \rangle$ (a detailed description is provided in Section 2.1.1), which constitute the set $\{R, P, L\}$.

We assume that for KB_S and KB_T, there exist *equivalences* between entity instances common in both KBs, and that are explicitly stated and materialized in the form of

`owl:sameAs` statements. For simplicity, we assume that each KB stores the statements indicating the equivalences between its instances and the instances of the other KB. More precisely, given two equivalent instance $e_s \equiv e_t$, the first from KB_S and the second from KB_T, we assume that, KB_S stores the fact `owl:sameAs`(e_s, e_t) and KB_T the fact `owl:sameAs`(e_t, e_s).

Definition 4.2.1 (Relation Subsumption)**.** *Given two relations r_S and r_T, we say that r_S is* subsumed *by r_T or r_T* subsumes *r_S and we write $r_S \subseteq r_T$ or $r_S \Rightarrow r_T$* iff

$$\forall x, y,\ r_S(x, y) \textit{ is } \text{true} \rightarrow r_T(x, y) \textit{ is also } \text{true}.$$

Example 10. *Given the following relations r_S = `dbp:locationCountry` and r_T = `yago:location`, we can establish that r_S is a sub-relation or that it is subsumed in r_T. This is because r_T represents and contains all information that can be encoded through r_S.*

Goal: The goal of our work is to solve the following problem: *Given two knowledge bases, a source $KB_S(R_S, P_S, L_T)$ and a target $KB_T(R_T, P_T, L_T)$, and a relation $r_S \in P_S$, find the relations $r_T \in P_T$ s.t. $r_S \subseteq r_T$.* We note that equivalence is a particular case of subsumption. Indeed, $r_S \equiv r_T$ *iff* $r_S \subseteq r_T$ and $r_T \subseteq r_S$. Hence, we support the computation of both relationships: subsumption and equivalence.

Note that according to our definition, the notion of subsumption is independent of the particular extensions of the two relations that are stored in some KBs. However, we can only rely on the facts stored in the KBs to learn subsumption relationships between relations. If the two KBs do not contain errors, and if indeed $r_S \subseteq r_T$ then

$$\forall x, x', y, y',\ \text{s.t. } x \equiv x' \wedge y \equiv y' \wedge r_S(x, y) \in KB_S \Rightarrow r_T(x', y') \in KB_T$$

For simplicity, in what follows we will use the same variable to denote an entity and its equivalent in another KB. In this work we consider only entity-entity relations, and leave the entity-literal relations as future work (see Definition 2.1.2).

4.3 Relation Alignment Model

In this section, we describe the *relation alignment* model, which we propose to find subsumption relation alignment between two KBs.

Consider again the knowledge bases $KB_S(R_S, P_S, L_T)$ and $KB_T(R_T, P_T, L_T)$ and an entity-entity relation $r_S \in P_S$. Our method is two fold:

(1) compute the set of super-relation candidates $C(r_S) = \{r_T \mid r_T \in P_T\}$

(2) for every pair $\langle r_S, r_T \rangle$ where $r_T \in C(r_S)$, check if $r_S \subseteq r_T$ and output *correct* if the relationship holds and *incorrect* otherwise based on a *supervised* machine learning model.

In the following, we describe the process on generating candidate relations from a target KB_T, namely $\langle r_S, r_T \rangle$ in Section 4.3.1. Next, we describe the individual features we compute for our alignment model in Section 4.3.2 and the learning framework in Section 4.3.3.

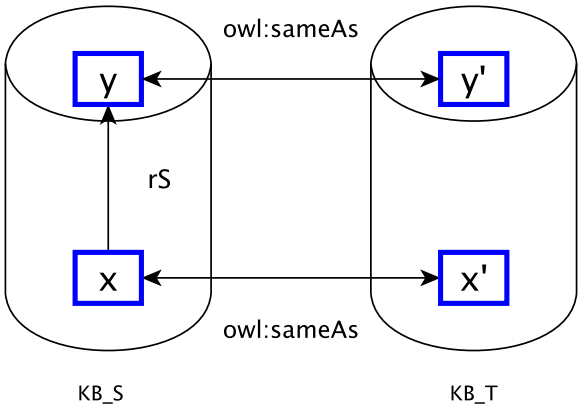


Figure 4.1: Selection of (x', y') entities constituting the S_{r_S} set.

4.3.1 Candidate Generation

Here we describe the process of generating the relation candidate pairs for alignment. For a given source relation r_S from P_S, we generate the tuple pairs $\langle r_S, r_T \rangle$ with $r_T \in P_T$.

Considering a relation r_S from our source KB_S. We collect entity pairs $(x, y) \in R_S^2$ for which r_S holds, that is, $\langle x, r_S, y \rangle \in KB_S$. We are interested only to those entity pairs for which the corresponding `owl:sameAs` statements exist and link them to our target KB_T. The entity instances are extracted from the corresponding SPARQL endpoint of the source and target KBs. The process of extracting entity pairs for r_S is depicted in Figure 4.1 and is defined in Equation 4.1.

$$S_{r_S} = \{(x', y') \in R_T^2 \mid \exists x, y \in R_S, x \equiv x', y \equiv y', r_S(x, y) \in KB_S\} \tag{4.1}$$

To extract the set of instances S_{r_S}, respectively the set of x', y' entities, we use the following query Q_1.

Q_1:
```
SELECT DISTINCT ?x' ?y' FROM <KB_S>
WHERE {  ?x      r_S      ?y.
         ?x owl:sameAs ?x'.
         ?y owl:sameAs ?y'.
}
```

Next, we use the S_{r_S}, respectively the entities (x', y') to generate candidate relations r_T from KB_T, such that r_T is a relation associated with (x', y'), $\langle x' \ r_T \ y' \rangle \in KB_T$ or $r_T(x', y') \in KB_T$. The set of candidate relations $r_T \in R_T$ is defined in Equation 4.2, the process is shown in Figure 4.2.

$$C(r_S) = C^+(r_S) \cup C^-(r_S) \tag{4.2}$$

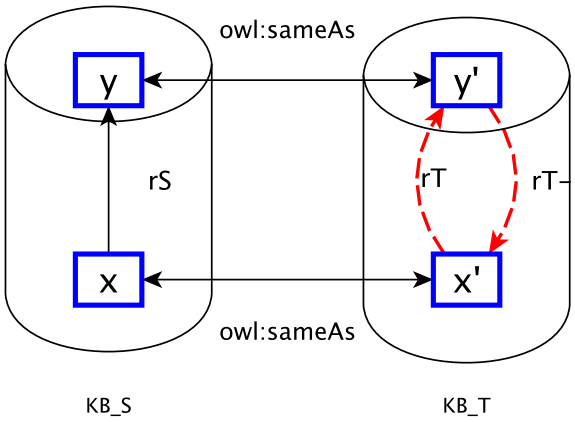


Figure 4.2: Relations in KB_T that hold between one or more instances of S_{r_S} set.

In the set of relations $C(r_S)$ we consider *direct* and *inverse* relations given by they following two sets:

$$C^{+}(r_S) = \{r_T \in P_T \mid \exists (x', y') \in S_{r_S} \wedge r_T(x', y') \in KB_T\} \tag{4.3}$$

$$C^{-}(r_S) = \{r_T \in P_T \mid \exists (x', y') \in S_{r_S} \wedge r_T(y', x') \in KB_T\} \tag{4.4}$$

The intuition of an *inverse* relation (Definition 2.1.3) is explained in the following example:

Example 11. *Consider the relation* $r_S =$ `dbp:author`*, specifically the triple*

⟨ dbp:Oliver_Twist `dbp:author` dbp:Charles_Dickens ⟩.

The relation in the KB_T *that encodes similar information and should be a candidate relation is* $r_T =$ `yago:created`*. Here is an example triple of the relation:*

⟨ yago:Charles_Dickens `yago:created` yago:Oliver_Twist ⟩.

It is obvious that we would not be able to uncover this r_T *if we would expect it to have the same* domain *and* range *as the* r_S*. But if we swap the domain and range then* `yago:created` *will be in the set with the candidate* r_T *relations.*

Therefore, we consider as relation candidates the inverse relations, in which case r_T^- would have the same *domain* and *range* definitions as r_S.

To construct the set $C(r_S)$, taking into account direct and inverse relations, we use the following SPARQL query Q_2. We issue such query against the target KB_T, and obtain the r_T relations for a specific S_{r_S} set of entities of a given relation r_S.

```
Q2:  SELECT ?r_T ?d FROM <KB_T> WHERE {
VALUES (?x' ?y') {
       x_1 y_1
       ...
       x_n y_n
}
{ SELECT ?x' ?r_T ?y' ?d WHERE{
       ?x' ?r_T ?y'.  VALUES ?d {"direct"}
}
UNION
{ SELECT ?x' ?r_T ?y' ?d WHERE{
       ?y' ?r_T ?x'.  VALUES ?d {"inverse"} }
}
}
```

We are not interested in the whole content of the KB_T, but only to those entities of r_S that are in a `owl:sameAs` relation to entities from KB_T a.k.a the entity instances of S_{r_S} set. We use the `VALUES` SPARQL construct to directly add the (x', y') values form S_{r_S} into the query, here the x_i and y_i represent real values. In this way we combine inline data with the result of the query execution by a `JOIN` operation. We use `UNION` to combine data from two queries. The two queries are to check both directions of the possible r_T, that is *direct* and *inverse* relations.

4.3.2 Features

In this section, we describe the process of computing features for a relation pair $\langle r_S, r_T \rangle$ (generated in the previous step), which we use to learn the relation alignment models to determine whether r_S is subsumed in r_T. We consider a set of light-weight features for the relation alignment process, grouped into three main categories. Firstly, we consider inductive logic programming (ILP) rule mining approaches, which work under the *open* and *closed* world assumptions. In the second group, we look into general relation statistics (GRS) for the pair $\langle r_S, r_T \rangle$, that specifically rely on the set of entity instances which are associated through r_S and r_T . Finally, in the third group, we consider the lexical similarity between the relation names from the pair $\langle r_S, r_T \rangle$. We present an overview of the feature list in Table 4.1.

In the following, we describe in details the individual features.

ILP – Features

Closed World Assumption – CWA. The first feature, cwa, is a measure that was proposed in [31]. It is used to mine association rules and was first used in a similar context for relation alignment between to KBs in [102]. The cwa measure works under the *closed world assumption*, where the KBs are assumed to be complete, hence there are no missing statements. Missing statements for an entity in two KBs are regarded as counter-examples. In cwa we measure the overlap in terms of the number of instantiations that

id	*feature*	*description*	*group*
f_1	*cwa*	closed world assumption	(ILP)
f_2	*pca*	partial completeness assumption	
f_3	*pia*	partial incompleteness assumption	
f_4	*func*	relation functionality	(GRS)
f_5	*WJ*	weighted jaccard similarity	
f_6	*WJD*	weighted jaccard relation dissimilarity	
f_7	$p(correct\|pca)$	prior probability of pca score	
f_8	$p(correct\|cwa)$	prior probability of cwa score	
f_9	$p(correct\|pca, cwa)$	joint likelihood of pca and cwa score for a relation pair being relevant	
f_{10}	$\lambda(r_S, r_T)$	lexical similarity for the relation pair $\langle r_S, r_T \rangle$	(Lexical)

Table 4.1: The set of computed features for the relation alignment model.

are common to r_S and r_T, normalised by the number of instantiations of r_S. Formally, the metric is defined, as following:

$$cwa(r_S \Rightarrow r_T) := \frac{\#(x,y) : r_S(x,y) \wedge r_T(x,y)}{\#(x,y) : r_S(x,y)} \tag{4.5}$$

where the number in the nominator $\#(x,y)$ corresponds to the number of pairs (x,y) that fulfill $r_S(x,y) \wedge r_T(x,y)$. Whereas, in the denominator, the number corresponds to all instantiations of $r_S(x,y)$. Note that the pairs (x,y) that are instantiations of r_S but not of r_T, are regarded as counter-examples for the rule $r_S \Rightarrow r_T$. To compute the cwa score we count the instances where the two relations overlap, and the counter-examples, through SPARQL queries on KB_S and KB_T.

Example 12. *Consider the candidate alignment:*

`yago:bornIn` ⇒ `dbpedia:birthPlace`

Assume that the KB_S contains the fact `yago:bornIn`*(Victor_Hugo, Besançon), but the KB_T does not contain the fact* `dbpedia:birthPlace`*(Victor_Hugo, Besançon); the instance (Victor_Hugo, Besançon) is regarded as a counter-example for the alignment. Hence, it contributes* 0 *to the numerator and* 1 *to the denominator of* cwa.

The cwa measure works well when the relations are *complete* (i.e., KBs have the complete set of statements for a given relation). However, this is not usually the case for KBs extracted from the Web, using information extraction algorithms. This for two

main reasons: (i) KBs are constructed from different sources of information (e.g. DBpedia is generated from Wikipedia, while YAGO is constructed by merging Wikipedia and WordNet), and (ii) entity co-references are not canonicalized to a global URI or are not linked explicitly through `owl:sameAs` statements, hence, not being able to interpret the different entity URIs as the same.

Partial Completeness Assumption – PCA. The second feature, pca, is an ILP measure proposed in [47] with the purpose of mining *Horn rules* for an input KB. The pca measure is able to handle incompleteness in KBs. It is designed to work under the *partial completeness assumption*. For a relation r and instance x, it is assumed that the KB contains either *all* or *none* of the r-triples having x as a *subject*. Note that a particular case that this assumption holds is the relations that are functions i.e. for which $func = 1$ (according to Equation 2.2). Under this assumption, we can count as counter-examples for the alignment $r_S \Rightarrow r_T$, any pair (x, y) that is an instantiation of r_S but not of r_T, more specifically for r_T we have statements where x exists as a subject but not y, that is, $y_2 \neq y$, respectively $r_T(x, y_2)$. The number of counter-examples is computed, as following:

$$counter(r_S \Rightarrow r_T) := \#(x, y) : r_S(x, y) \wedge \exists y_2, y_2 \neq y : r_T(x, y_2) \wedge \neg r_T(x, y) \quad (4.6)$$

This yields the following confidence measure:

$$pca(r_S \Rightarrow r_T) := \frac{\#(x, y) : r_S(x, y) \wedge r_T(x, y)}{\#(x, y) : \exists y_2 : r_S(x, y) \wedge r_T(x, y_2)} \in [0, 1] \quad (4.7)$$

Like for the cwa, the numerator is the number of instantiations shared in common by r_S and r_T:

$$overlap(r_S \wedge r_T) := \#(x, y) : r_S(x, y) \wedge r_T(x, y) \quad (4.8)$$

However, the denominator represents the sum between the number of counter-examples and the overlap. Hence, pca can be also written as follows:

$$pca(r_S \Rightarrow r_T) := \frac{overlap(r_S \wedge r_T)}{overlap(r_S \wedge r_T) + counter(r_S \Rightarrow r_T)} \quad (4.9)$$

To compute the pca score we count the instances where the two relations overlap according to the Equation 4.8, and the counter-examples according to the Equation 4.6, through SPARQL queries on KB_S and KB_T.

Note that unlike cwa, pca measure does not penalize the relation alignment pair if the target KB_T does not store for some instance (x, y) of r_S any r_T fact for x.

Example 13. *Consider again the Example 12. The instance (Victor_Hugo, Besançon) is regarded as a counter-example only if the target KB contains a fact* `dbpedia:birthPlace` *(Victor_Hugo, Y) other than* `dbpedia:birthPlace`*(Victor_Hugo, Besançon).*

We remark that pca can have maximal score even when the overlap between the instantiations of the two relations consists of few tuples with no counter-examples. This is helpful in detecting alignments with a small overlap in the two KBs. On the other hand, it increases the likelihood of getting false positives. These are caused by erroneous facts, incomplete data and overlapping relations.

However, one way to circumvent the shortcomings of cwa and pca is considering them jointly as features. This has the effect of *regularizing* each other, where for high pca score but low cwa score the chance of having a correct alignment is low, and vice-versa. We will make use of this assumption and compute a *joint likelihood* score for a relation pair being correct by considering the scores of pca and cwa.

Partial Incompleteness Assumption – PIA. The partial completeness assumption may hold for *functional relations* (see definition of relation functionality in Section 2.1.1), but is less probable for one-to-many relations. Indeed, the intuition is that if the average number of r_S-triples per subject is high, then it is more likely that not all the r_S–triples of some subject x have been extracted. The same observation holds also for the triples of r_T. Note, that we have yet to prove that r_S is subsumed in r_T. Hence, the counter-examples of less functional relations should be weight less than the counter-examples of more functional relations. To this end, we propose the pia score, defined as follows:

$$pia(r_S \Rightarrow r_T) := \frac{overlap(r_S \wedge r_T)}{overlap(r_S \wedge r_T) + (counter(r_S \Rightarrow r_T) \times func(r_S))} \tag{4.10}$$

where, $func(r_S)$ measured as the probability that a counter example may not exist.

General Relation Statistics (GRS) – Features

The features in the ILP group are concerned with entity statements for any two relations under consideration. Here, in the *general relation statistics* feature group, we propose features that take into account the cardinality of the two relations, and their *domains* and *ranges*, correspondingly the entity types extracted from entity instances in the respective domains and ranges of a relation.

Functionality of Relations. The first feature in the GRS feature group is the *relation functionality*. We consider the correlation of the relation functionalities (see Equation 2.2) from the pair $\langle r_S, r_T \rangle$.

A relation is considered to be highly functional, if for a subject entity we have one object entity. On the other hand, when the functionality of a relation is very small, for a given subject entity the relation has many object entities.

Following this intuition, we assume that if a relation r_S is subsumed by a relation r_T then the number of r_S facts per subject entity should be lower than the number of r_T facts per subject entity. In other worlds, the functionality of r_S should be grater or equal to the functionality of r_T. Therefore, we desire the following relationships between the relation

functionalities r_S and r_T:

$$if\, r_S \Rightarrow r_T \rightarrow func(r_S) \geqslant func(r_T)$$

The feature that we consider in this setting is the difference of the two functionalities.

$$func_{diff} := func(r_S) - func(r_T) \in (-1, 1)$$

Weighted Jaccard Similarity. In this feature, we measure the similarity of the entity type distributions $\mathscr{D}(\cdot)$ between the relation pairs $\langle r_S, r_T \rangle$.

We gather such statistics from the r_S and r_T concerning the *entity type* distribution, specifically the type distributions of *subjects* entities, in other words the type distribution of the domains of the relations. We denote the following relation statistics, $\mathscr{D}^s(r_S)$ and $\mathscr{D}^s(r_T)$, as the type distribution for entities as *subject* values for relations r_S and r_T, respectively.

The distributions represent a set of entity types, where for each entity type we have the proportion of entities (out of the total entities for a relation) that belong to that type. For example, $\mathscr{D}^s(r_S) = \{\langle type_1, 0.5\rangle, \langle type_2, 0.2\rangle, \ldots\}$.

Since the two KB that we consider use different schemas, for uniformity, we pick one of the two type taxonomies used to represent the entity instances and compute the statistics for the types distribution. We are able to bring the entity type representation into one specific type taxonomy due to our restriction where we consider only entities that are explicitly linked across KBs through `owl:sameAs` statements. We provide a detailed description in our experimental evaluation regarding the choice of KBs and the taxonomies used for this feature in Section 4.5.2.

For the time being, lets assume we pick the type taxonomy of the KB_S to represent the types for both r_S and r_T. For r_S we have:

$$r_S(x, \cdot) \wedge \texttt{rdf:type}(x, type_1) \rightarrow type_1 \in \mathscr{D}^s(r_S) \tag{4.11}$$

For r_T we have:

$$r_T(x', \cdot) \wedge x' \equiv x : x \in R_S \wedge \texttt{rdf:type}(x, type_1) \rightarrow type_1 \in \mathscr{D}^s(r_T) \tag{4.12}$$

Considering that items in the respective entity type distributions are scored based on the proportions of entities belonging to that type for a relation and KB. In this case we can use several similarity measures (e.g. *cosine, weight jaccard*, we opt for *weighted jaccard* as a similarity metric without loss of generality.

$$WJ(\mathscr{D}(r_S), \mathscr{D}(r_T)) = \frac{\sum\limits_{t \in \mathscr{D}(r_S) \wedge \mathscr{D}(r_T)} min(\sigma(t^S), \sigma(t^T))}{\sum\limits_{t \in \mathscr{D}(r_S) \wedge \mathscr{D}(r_T)} max(\sigma(t^S), \sigma(t^T))} \tag{4.13}$$

where, $\mathscr{D}(\cdot)$ represents either the type distribution for the subject or object values based on that if the relation is a direct or inverse relation. The function $\sigma(t)$ represents the score of an entity type in the distribution for a given relation.

The intuition behind the *weighted jaccard* similarity is the following. Due to the fact that we want to find if r_S is subsumed in a target relation r_T, the respective type distributions $\mathscr{D}(r_S)$ and $\mathscr{D}(r_T)$, should be similar, or $\mathscr{D}(r_S)$ should be entailed in $\mathscr{D}(r_T)$. In other words, the target relation r_T, should be able to represent entity instances from r_S, namely through its *domain* entity type definition [3].

This feature is best explained through a concrete example.

Example 14. *Consider the following distributions:*
$\mathscr{D}(r_S) = \{\langle \texttt{Movie}, 0.8\rangle, \langle \texttt{Book}, 0.1\rangle, \ldots\}$,
$\mathscr{D}(r_T) = \{\langle \texttt{Book}, 0.8\rangle, \langle \texttt{Movie}, 0.05\rangle, \ldots\}$.
The corresponding WJ score would be low, that is, relation r_T is unlikely to subsume r_S due to the fact that based on its entity type representation, type `Movie` *represents only 5% of the total instances while in r_S is the 80% of the total instances.*

Therefore, we assume, that for a relation to subsume another, the respective type distribution need to be closely similar. Finally, by taking into consideration the proportion of entity instances belonging to a type, in this feature we can compare relations that have varying number of entity instances (due to the inherent size of a KB).

Weighted Jaccard Dissimilarity. Similarly as for the *weighted jaccard similarity*, which is computed on the overlapping types between the two entity type distributions $\mathscr{D}(r_S)$ and $\mathscr{D}(r_T)$, in this case, we compute the *weighted dissimilarity score* (WDS) between r_S and r_T. Specifically, if a specific entity type from r_S does not exist in r_T, this accounts for a dissimilarity between the two relations, and lowers the likelihood of r_T subsuming r_S. The WDS score is computed as following.

$$WDS(\mathscr{D}(r_S), \mathscr{D}(r_T)) = \frac{\sum_{t \in \mathscr{D}(r_S) \wedge t \notin \mathscr{D}(r_T)} \sigma(t^S)}{|\mathscr{D}(r_S) \setminus \mathscr{D}(r_T)|} \tag{4.14}$$

The intuition behind this feature is that even if the WJ score is high (computed on the overlapping types), in case r_T cannot represent specific entity types from r_S, those account for a dissimilarity score. Respectively, the chance of r_S being subsumed by r_T is decreased. The higher the WDS score is the less possible is the relationship $r_S \Rightarrow r_T$ to hold.

Example 15. *Consider the following distributions:*
$\mathscr{D}(r_S) = \{\langle \texttt{Movie}, 0.3\rangle, \langle \texttt{Book}, 0.2\rangle, \langle \texttt{Painting}, 0.5\rangle \ldots\}$,
$\mathscr{D}(r_T) = \{\langle \texttt{Book}, 0.2\rangle, \langle \texttt{Movie}, 0.3\rangle, \ldots\}$
In the case where the common types of the relation pairs have similar distribution, the WJ score will be high. However, if there is a type in the r_S namely, `Painting` *which does not exist in r_T, that is not taken into account in the computation of WJ. This type represents the 50% of the entities in r_S. That means, the relation r_T is unlikely to subsume r_S and this is discovered by the WDS score which is going to have the value WDS = 0.5*

3. Each relation in a RDFS schema has two properties denoting the *domain* and *range*.

ILP Score Relevance Likelihood. Recent work and survey on *ontology matching* approaches [99] has shown that dependent on the dataset pair under consideration, the choice of *matchers* varies, in our case, the features we compute for our relation alignment model. Furthermore, the computed values for the individual features are dependent on the KB pair.

This is especially true for the ILP features. If we consider pca or cwa, a scores that is not nearly perfect is likely to be a good indicator for subsumption if the overlap in terms of statements is high in absolute numbers. Yet, we can achieve high scores even with a low overlap of statements in absolute numbers, hence, as such it will not guarantee the subsumption of a relation in target KB.

For the first case, if we have a score of $pca = 0.8$, with a high number of overlapping statements, that is the set of (x, y) represents a large number of entities being in common and having sufficiently complete set of facts. Such a score is a strong indicator of subsumption. Now, consider the second case where we get a score of $pca = 1.0$, for (x, y) pairs, the relations overlap completely, however, the statements come from a single entity x, such a score will be unlikely to guarantee a relation subsumption.

Therefore, in this feature group, we assess the likelihood of the specific scores being indicators of subsumption for a relation pair. It is obvious that the probability distributions are measured only on a *training sample* of relation alignment candidates, whereas for testing instances we simply asses their scores against the computed probability distributions.

For this purpose, we discretize the computed pca and cwa score into the ranges with a cut-off point of 0.1, $\{0, 0.1, \ldots, 1.0\}$, and compute the likelihood of a relation alignment being correct given a specific pca, cwa or the joint pca and cwa score.

We will denote with $\langle r_S, r_T \rangle^c$ the set of relation alignment candidates that are correct, and with $\langle r_S, r_T \rangle$ the complete set of relation alignment candidates.

PCA Prior. Here, we compute simply the *priors* of a specific pca score, from the aforementioned range, respectively for which the pair $\langle r_S, r_T \rangle^c$. The pca prior is computed as following.

$$p(correct|pca) = \frac{\#\langle r_S, r_T \rangle^c : pca(r_S \Rightarrow r_T) = pca}{\#\langle r_S, r_T \rangle : pca(r_S \Rightarrow r_T) = pca} \tag{4.15}$$

In the numerator we count the $\#\langle r_S, r_T \rangle^c$ pairs that are correct for a given pca score. Where in the denominator we count all the pairs $\langle r_S, r_T \rangle$ (correct or incorrect) that have the given pca score. In summary, the $p(correct|pca)$ simply measures the ratio of the number of cases where for a given discretized pca score the subsumption alignment holds divided by the total relation pairs having a given pca score. Note that, we learn these probabilities on a given set of training instances (see Section 4.5).

CWA Prior. Similarly as for the computed probability pca score, we compute a the $p(correct|cwa)$ score for the discretized cwa scores. The computation and intuition is the same as for $p(correct|pca)$, we compute $p(correct|cwa)$ as following in Equation 4.16.

$$p(correct|cwa) = \frac{\#\langle r_S, r_T \rangle^c : cwa(r_S \Rightarrow r_T) = cwa}{\#\langle r_S, r_T \rangle : cwa(r_S \Rightarrow r_T) = cwa} \tag{4.16}$$

Joint PCA & CWA Likelihood. As mentioned in the beginning of this section, there are advantages and disadvantages from the pca and cwa scores, respectively. In this measure, we compute the *joint likelihood* that for a given pca and cwa score, the relation pair $\langle r_S, r_T \rangle$ is correct. Similarly, we first discretized the scores and compute the joint relevance likelihood as in Equation 4.17.

$$p(correct|pca, cwa) = \frac{\#\langle r_S, r_T \rangle^c : pca(r_S \Rightarrow r_T) = pca \wedge cwa(r_S \Rightarrow r_T) = cwa}{\#\langle r_S, r_T \rangle : pca(r_S \Rightarrow r_T) = pca \wedge cwa(r_S \Rightarrow r_T) = cwa} \tag{4.17}$$

where for a given pca and cwa score we simply count the number of relation pairs whose scores match and the corresponding alignment is relevant, over the total number of relation pairs with the respective pca and cwa scores. We expect this feature to be sparse, hence, we use the simple priors as fall-back features.

Lexical Relation Similarity

In this feature group, we simply measure the *lexical similarity* between the relation names for the pair $\langle r_S, r_T \rangle$. This presents a standard feature used in ontology matching approaches [99]. In order to compute the lexical similarity features, we remove the base URL of each relation and split the relation names based on the capitalization of the relation name letters, i.e., `hasPrize`={`has, Prize`}. Finally, we lowercase all the resulting tokens.

The score of the feature $\lambda(r_S, r_T)$ corresponds simply to the *Levenshtein* [66] distance between the relation names.

It is worth noting that, in this feature group one can use external dictionaries like WordNet, to map the individual terms in a relation to a multidimensional vector of possible synsets (e.g. `Prize` = {`prize, award, trophy, ...`}). This allows for more expressive representation of the relation names, hence, the computed similarity will be more representative. However, we leave such implementation for future work and consider the simple lexical similarity.

Finally, from all the computed features, we train supervised machine learning models, who combine the individual features into a model which we use to assess the relevance of the proposed subsumption relation alignments.

4.3.3 Relation Alignment Supervised Models

In this section, we provide an intuition behind our decision on combining the afore mentioned features. As presented above as part of our intuition, the individual features, which serve as matchers for our relation alignment problem do not yield optimal performance when applied separately. For example, if we consider the *ILP* feature group, namely, the cwa and pca scores, if applied separately, they are either too *restrictive* (in the case of cwa measure) causing too many false negatives, or too *relaxed* (pca measure) allowing for too many false positive relation alignments. Hence, the intuitive approach

here is to combine the features in a supervised machine learning model. Through supervised models, we learn automatically the importance of features and as a consequence are able to predict accurately the label for the pairs $\langle r_S, r_T \rangle \rightarrow \{correct, incorrect\}$.

One minor drawback of using supervised models is the need for *ground-truth* data. In order to train such models, we need to create ground-truth data for which we know the correct label for the relation pairs, that is, $\langle r_S, r_T \rangle$, which we use as *training data*. However, as we will see later in our experimental evaluation in Section 4.6, the amount of training data we need for our models to converge is minimal. In Section 4.5.3 we discuss approaches and how we construct our ground-truth for learning the relation alignment models.

In the following, we describe two main aspects of our model, namely, the choice of the *supervised models* and a *feature selection* algorithm, which we use to select a subset of features used for training.

Supervised Models

We propose a non-exhaustive list of possible supervised models that can be trained to predict the relevance of a relation alignment for a given pair $\langle r_S, r_T \rangle$. We learn a binary classification model where we distinguish between *correct* and *incorrect* relation alignment pairs.

We mainly focus on models whose complexity is linear, $\mathcal{O}(n)$. We hypothesize that a linear combination of our features places the relation pairs in a *linearly separable* space. Therefore, in the following we discuss two well known, linear supervised models that we use in our approach and later on evaluate in Section 4.5. For both models used in this work we are based on the implementation provided by Weka [51], a widely used machine learning framework.

Voted Perceptron. The first model we consider for learning the relation alignments is the *Voted Perceptron* (VP) algorithm [43]. It combines features into a linear model by weighing them based on the impact they have on predicting accurately the label of the candidate pair $\langle r_S, r_T \rangle$.

The VP algorithm is based on the *Perceptron* (PC) algorithm. The objective of the VP algorithm (this holds for the PC algorithm too) is to find an optimal *set of weights* for the individual features such that the number of miss-classifications is minimized.

What sets VP apart from PC is that it stores additional information during the learning process. In VP the model keeps a *set of feature weight sets* (in PC a feature is weighted by one specific value), more specifically, the individual weights from this set of feature weights are weighted proportionally to the number of correct predictions.

In more details, for a given set of training instances and a maximum number of iterations the following steps are undertaken. Lets denote the set of training instances $\{(\langle r_S^1, r_T^1 \rangle, y_1), \ldots, (\langle r_S^m, r_T^m \rangle, y_m)\}$. Each training instance $\langle r_S, r_T \rangle$ is constituted by the individual feature values $f(\langle r_S, r_T \rangle)$ (see Table 4.1 for the complete set of features), and the corresponding label $y \in \{+1, -1\}$ for *correct* and *incorrect* alignment respectively.

In the initialization step of the VP algorithm, the weights of the individual features are assigned to a random value between $[0, 1]$ or zero, $\mathbf{w} = \mathbf{0}$.

Next, for each individual training instance $\langle r_S, r_T \rangle$, the VP model with the current state of the feature weights $\mathbf{w}$ predicts the label of the instance as $\hat{y} = \text{sign}(\mathbf{w} \cdot f(\langle r_S, r_T \rangle))$. In case the function $\hat{y}$ under the current state of weights $\mathbf{w}$ predicts the wrong label, then the weights are adjusted according to $\mathbf{w} = \mathbf{w} + y \cdot \alpha \cdot \mathbf{f}(\langle \mathbf{r_S}, \mathbf{r_T} \rangle)$. In more details, the individual weights for each feature are adjusted by adding or subtracting the feature value multiplied with a pre-specified *learning rate* α (usually a value $\alpha < 1.0$), if we miss-classify the positive or negative class, respectively. Here with $f(\langle r_S, r_T \rangle)$ we denote the specific feature value for the training instance $\langle r_S, r_T \rangle$. Finally, the weights $\mathbf{w}$ are not changed in case the prediction by $\hat{y}$ corresponds to the real label of a training instance.

As mentioned in the beginning of this section, the VP algorithm keeps a set of weight sets $\mathbf{W} = \{\mathbf{w_1}, \ldots, \mathbf{w_n}\}$, and each weight set has a score c that corresponds the number of correctly labeled instances.

The VP algorithm halts when the learning process has converged, that is, when the model does not miss-classify any of the training instances, or in the case where the maximal number of iterations is reached.

Finally, from the set of all weighted *feature weights*, we predict the label of a relation pair through a weighted majority voting, with votes having higher weights proportional to the *feature weights* (weights assigned based on the number of instances they predict accurately). The classification model is formally defined in Equation 4.18.

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{k} c_i * sign(w_i \cdot f(\langle r_S, r_T \rangle))\right) \tag{4.18}$$

Logistic Regression Model. *Logistic Regression* (LR) represents a linear supervised model [19]. Given our training instances and the respective feature values $f(\langle r_S, r_T \rangle)$, we learn the function $\hat{y}_{correct}$ (for a binary classification task, the *incorrect* class is computed as $1 - \hat{y}_{correct}$). The advantage of the *logistic model* is that it is a linear expression, and thus, the achieved prediction labels are easily explainable. The model is computed as in Equation 4.21, where we show the likelihood of the relation pair $\langle r_S, r_T \rangle$ being a *correct* alignment. The other case is the pair being an incorrect alignment, $\hat{y}_{incorrect}$, can be simply computed through $1 - \hat{y}_{correct}$.

In LR, the prediction of the labels, namely the functions, $\hat{y}_{correct}$ and $\hat{y}_{incorrect}$, can be represented as conditional probabilities, specifically, $P(y = 0 | f(\langle r_S, r_T \rangle)$ and $P(y = 1 | f(\langle r_S, r_T \rangle)$, respectively. It simply states that under the LR model what is the likelihood that for a given set of feature values one of the labels to hold, i.e. $y = 1$.

In the LR, the model that is used to compute the likelihood of a label given the training data, we first need to estimate the parameters $\mathbf{w}$, which represents feature weights from the feature set $f(\langle r_S, r_T \rangle)$. The weights are estimated such that for a given set of training instances, the LR model is optimal and thus the number of miss-classifications is minimal. A standard approach to achieve this is through maximum likelihood principle (MLP) (see Equation 4.19). In MLP, the parameter W is considered a free parameter for which we

find values that maximize the performance of the LR model, that is, the number of miss-classification is minimal.

$$W \leftarrow \arg\max_{W} \prod_{i} P(y_i | f(\langle r_S, r_T \rangle), W) \tag{4.19}$$

From the Equation 4.19 we can deduce a function $\mathscr{L}(W)$ in Equation 4.20 which we use to estimate the weights. A common approach to obtain weights that maximize the likelihood function in Equation 4.19 is through *gradient ascent*. The principle of gradient ascent is similar to the adjustments of weights in the VP model. It iteratively adjusts the weights for a given set of training instances. Detailed analysis of the gradient ascent parameter estimation approach is provided in [19], where it is shown that the weights computed for $\mathscr{L}(W)$ are optimal and thus maximize the likelihood principle in Equation 4.19.

$$\mathscr{L}(W) = \prod_{i} \big(y_i P(y_i = 1 | f(\langle r_S, r_T \rangle), W) + (1 - y_i) P(y_i = 0 | f(\langle r_S, r_T \rangle), W) \big) \tag{4.20}$$

Finally, after estimating the weights W for the individual features, we can predict the label of relation alignment pairs through the Equation 4.21. It must be noted that the label of an instance is determined based on which score is higher, that is $\hat{y}_{correct} > \hat{y}_{incorrect}$, for *correct*, and vice-versa for *incorrect*.

$$\hat{y}_{correct} = \frac{1}{1 + exp\left(w_0 + \sum_{i=0}^{n} w_i * f(\langle r_S, r_T \rangle)_i \right)} \tag{4.21}$$

Feature Selection

An important aspect that we consider for our supervised model, is the choice of features we use for training such models. As evidenced in related work [99], dependent on the datasets, the choice of features (*matchers* as referred in the *ontology matching* literature) that would have optimal performance will vary. Hence, we consider *feature selection* when learning our relation alignment models.

The advantages of *feature selection* are two fold: (i) through feature selection we choose the features that optimize our model on predicting accurately the relation pairs under consideration, and (ii) our models are more generalizable by choosing only the top-performing features, hence, avoiding over-fitting to the training instances.

Here we present one approach we use for feature selection that is based on *information gain* (IG) measures [127]. The IG feature selection approach measures the information gain we get from a specific feature in our feature set $f(\langle r_S, r_T \rangle)$ in predicting the label of the relation pair, i.e. $\langle r_S, r_T \rangle$. The computation of IG for a feature is done as following.

$$IG(f_i) = H(Y) - H(Y|f_i) = -\sum_{y \in \{-1,+1\}} P(y) \log P(y) + \sum_{k=1}^{n} \sum_{y \in \{-1,+1\}} P(y|f_i^k) \log P(y|f_i^k) \tag{4.22}$$

where $IG(f_i)$ measures the information gain for feature f_i, where $f_i = \{f_i^1, f_i^2, \ldots, f_i^n\}$ is a set of discretized feature values for f_i for a given set of training instances. In our case, the values correspond to scores as computed by the individual features in Table 4.1.

The gain takes into account the individual feature values and the information gain for each value for a given class, which is represented as a *conditional probability* $P(y|f_i^k)$ on predicting y given f_i^k. The score $P(y|f_i^k)$ simply measures if the specific feature value is a good indicator on predicting the labels $y \in \{-1, +1\}$ in our training set. The IG score is computed for each individual feature.

We compute the step of feature selection before learning our supervised models. The feature selection, namely the IG scores are computed on the training instances. We experiment with different cut-offs of top–k features and find the optimal set of features, namely, a small subset of features for which our models will be optimal in predicting correctly the labels of the relation alignment pairs.

4.4 Online Relation Alignment

In the previous section we described the proposed approach for relation alignment. However, as argued in Section 4.1, the ever increasing number of available linked datasets, and their evolving nature of the data make such a problem inefficient when considering approaches that require full access of the datasets. In many cases these datasets are available only through a SPARQL endpoint, thus, providing only a subset of data by placing limitations in the number of triples one can query at each SPARQL request. Furthermore, even when full access is provided, downloading and analyzing the full data is expensive (in terms of time and used network bandwidth) and does not scale. Lastly, alignments computed on a specific dataset snapshot might not be valid in another version.

Here we suggest *sampling* strategies which aim at addressing two main issues: (i) overcome the expensive step where the full dataset is required, and (ii) sample a minimal and representative set of instances that are used to build our relation alignment models, specifically compute the features. The sampling strategies take into account the nature of the datasets, respectively their set of relations, consequentially the sampled instances aim at maximizing the evidence that is provided for determining the correct label for a pair, and additionally being representative of the general population of entities that are associated with a specific relation.

We refer the reader to the problem description in Section 4.2, and in the following we introduce three sampling strategies described below.

4.4.1 Sampling Strategies

We propose three sampling strategies for entity instance selection, which we use to generate *relation alignment candidates*. Given our source KB_S and query relation r_S the main objective is to sample for a minimal set of entity instances S_{r_S}. For example, a relation that has as a domain *narrow concepts* (e.g. `Politician`), representing very homogeneous information, the assumption is that a small sample size is required. While

for relations that have as a domain *broad concepts* (e.g. `Person`), then we need to sample for a larger number of instances.

In summary, the sampling step in our approach produces the set of entity instances S_{r_S}, which contains a set of *representative* entity instances, while keeping a *minimal overhead* during the query–execution, in terms of *time* and *network bandwidth* usage. Moreover, through S_{r_S} we ensure that we have high *coverage* of alignments for r_S in a target knowledge base KB_T.

First–N Sampling

The first sampling strategy is first–N, which we use to extract samples in S_{r_S}, namely the entities (x', y') for relation r_S in KB_S. This represents a highly efficient query, it introduces a minimal overhead in the query-execution time. We execute the following SPARQL query Q_3, which returns the first–N entity samples (x', y').

Q_3:
```
SELECT DISTINCT ?x' ?y'
WHERE { ?x      r_S      ?y.
        ?x owl:sameAs ?x'.
        ?y owl:sameAs ?y'.
} LIMIT N
```

One drawback of selecting first–N entity samples for S_{r_S} as shown in Q_3, is that it violates the assumptions of constructing a *representative sample*. Query Q_3 is subject to factors that do not conform to a controlled environment. For instance, we might end up with a skewed sample distribution based on how the triples are added into a KB.

Another major issue with samples extracted through the first–N approach, is that in the samples (x', y'), x' might not be distinct in S_{r_S}. That is, in case we sample for 100 statements, and each statement belongs to a single entity x', we end up with very biased and not representative sample. Such samples are not representative, and the relation alignment model built upon such samples will have low coverage and is likely to overfit because of two main reasons.

First, for samples (x', y') in KB_S and KB_T and for a common x', the set of statements in both KBs might be complementary, that is, the features we compute for our models will assign low scores for the relation alignment pair. Second, in the case where (x', y') overlap completely in both KBs, the model would overfit to the specific x', hence failing to generalize and similarly such scores would not be representative for a larger sample of x'.

Yet, this represents a baseline to show the impact of sampling strategies on generating accurate relation alignments.

Random Sampling

A natural way to construct representative samples for S_{r_S} is to randomly sample over entities associated with relation r_S in KB_S. SPARQL 1.1 provides support for the function

RAND() [4] which sorts the matching triples in a random order. It produces a different random sample every time the RAND() function is invoked. We execute query Q_4 to get a random sample of N entity pairs (x', y') associated with r_S.

Q_4:
```
SELECT DISTINCT ?x' ?y'
WHERE { ?x      r_S      ?y.
        ?x owl:sameAs ?x'.
        ?y owl:sameAs ?y'.
} ORDER BY RAND() LIMIT N
```

Through random sampling we ensure that S_{r_S} consist of representative sample of entities (x', y'). However, ordering the matching triples through the RAND() function is expensive and introduces a significantly higher overhead in the query-execution time when compared to the first–N query Q_3.

Another issue one can encounter on constructing S_{r_S} through Q_4 is that in case we are dealing with a relation that has a range of broad concepts, then our samples will be biased towards the entities belonging to the more abstract classes. For example, the relation r_S =creatorOf can have as a domain entities of type Artist, Architect, Writer etc. Depending on the sample size, the finer grained entity types might not be included in our samples, therefore, having an impact on the possible candidate relations that we can align in KB_T which are specific to those types.

Stratified Sampling

To account for the possibility of missing candidate relations from KB_T, due to their domain/range being specialized entity types, i.e., Illinois_lawyers (an entity type associated with Barack_Obama), we employ our third sampling strategy, namely the *stratified sampling* technique [28]. Here, the main objective is to have better coverage in terms of uncovered relation alignments by taking into account the entity types associated with the subject/objects of a relation r_S.

The idea of stratified sampling is to divide members of the population into homogeneous groups (called *strata*) before sampling. In our case a strata is defined by the entity types from the subject entities x' associated with relation r_S. We extract the entity type of x' from the triple $\langle x'$ rdf:type Politician$\rangle$. For instance, from x', we can infer the coarser grained types from a type taxonomies present in a KB, e.g. $\langle x'$ rdf:type Person$\rangle$.

In this way we can express the *strata* based on types at different levels in the taxonomy. The higher in the taxonomy, our stratified sampling will be similar to the random sampling, whereas the deeper we go in the taxonomy, we will have higher number of strata, hence, increasing the coverage of aligned relations in KB_T. By varying the depth level in a type taxonomy we can find the optimal depth level such that the coverage in

4. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

terms of r_T is maximal. The intuition is that the generated samples (x, y) from r_S contain very representative instances from *all* possible entity types, hence, maximizing the coverage and accuracy of the discovered candidates r_T.

The size of the sample in each stratum is taken in proportion to the size of the stratum. This is called proportional allocation and we compute the size N of each *strata* as

$$N = \#stratum * \frac{\#sample}{\#totalStrataSize} \tag{4.23}$$

where $\#stratum$ is the number of subject entities that are instances of the specific entity type in a specific type taxonomy, $\#sample$ is our desired number of entities we want to sample, and $\#totalStrata$ is the number of subject entities belonging to all entity types for the relation r_S. Finally, we sample for (x', y') for the different *strata* as in query Q_5.

Q_5:
```
SELECT DISTINCT ?x' y'
WHERE { ?x      r_S      ?y.
        ?x owl:sameAs ?x'.
        ?y owl:sameAs ?y'.
        ?x  rdf:type  ?type.
} ORDER BY RAND() LIMIT N
```

Since our *strata* consist of entity types, and considering that the entities are associated with types and their *transitive closure*, this way of defining the strata violates one of the first assumptions of *stratified sampling* of disjoint items.

For example, for an entity ⟨*e* `rdf:type Actor`⟩, entity *e* will be part of all the strata that correspond to the transitive closure of `Actor`, i.e ⟨`Actor subClassOf Person`⟩ → ⟨`Person subClassOf Human_Being`⟩. However, by constructing the strata at specific depth levels in the type taxonomy we can associate an entity to its most specialized type, thus, ensuring *disjointness*.

4.5 Experimental Setup

In this section we present the experimental evaluation of our proposed approach for relation alignment. Our evaluation was carried on the following computing infrastructure. We implement our approach in Java (JVM 1.7), for hosting of our evaluation datasets we rely on the Virtuoso Universal Server[5] with SPARQL 1.1 support. All our experiments are carried on a server with Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20GHz and 50GB of main memory.

In the following subsections we describe in details the specific points in our experimental setup.

5. `http://virtuoso.openlinksw.com`

4.5.1 Datasets

We evaluate the proposed relation alignment approach on three real-world knowledge bases, YAGO [103], DBpedia [13], and Freebase [6]. We host these individual KBs in our RDF triple-store (Virtuoso). These represent the most common and well-established KBs. Further, they represent a real-world scenario as they are the most commonly used KBs and are general purpose datasets.

YAGO (Y). From the YAGO2 dataset we use the *core* facts [7], excluding the entity labels. We exclude the labels given that we consider only entity–entity relations. Additionally, we have the entity type information, where we use the transitive types. This amounts to approximately 900MB of triples.

DBpedia (D). In the case of DBpedia, we use the English version 3.9 [8]. Specifically, we consider the entity types (here too we have the transitive closure of types) and the subset of DBpedia, namely the mapping-based properties [9]. This amounts to 5.5GB of triples.

Freebase (F). Due to the large size of Freebase, we take a subset of Freebase dataset [10], that corresponds to entities that have `owl:sameAs` links to DBpedia. In the end, this amounts to 30GB of data.

Finally, from the aforementioned KBs, we extract all possible `entity-entity` relations, r_{ee}. We filter out relations that do not have more than 50 triples [11]. Table 4.2 presents some of the statistics for our datasets. In the first column is shown the total number of relations for each KB, whereas in the second column is shown the number of relations after filtering. In the third column we show the total number of triples in our datasets.

	relations		**$\geq$ 50 triples**		**#triples**
	r_{ee}	r_{el}	r_{ee}	r_{el}	
YAGO	37	41	36	38	139,630,953
DBpedia	688	690	563	565	64,600,376
Freebase	5855	2910	1666	861	220,339,399

Table 4.2: Statistics for the individual KBs, number of relations and the total number of triples.

Entity Links `owl:sameAs`: Since in this work we focus on `entity-entity` r_{ee} relation alignment, we make use of the `owl:sameAs` links between entities across the

6. `http://www.freebase.com`
7. The groups as indicated on the YAGO download page.
8. `http://wiki.dbpedia.org/services-resources/datasets/data-set-39/downloads-39`
9. Mapping-based properties in DBpedia represent statements about entities that are extracted from Wikipedia Infoboxes.
10. `https://developers.google.com/freebase/data`
11. We exclude relations below this threshold since they are not suitable for learning and cannot be included in the sampling process.

three KBs. Since in our datasets we have only owl:sameAs links between the pairs DBpedia and YAGO, and DBpedia and Freebase, we infer the owl:sameAs links between YAGO and Freebase through the DBpedia links. That is, for two entities in YAGO and Freebase we can infer the owl:sameAs link between them through the following transitivity rule.

$$e_Y \texttt{ owl:sameAs } e_D \wedge e_D \texttt{ owl:sameAs } e_F \longrightarrow e_Y \texttt{ owl:sameAs } e_F$$

Table 4.3 shows the number of owl:sameAs links between entities for the different KB pairs.

KB	YAGO – DBpedia	YAGO – Freebase	DBpedia – Freebase
#owl:sameAs	2,886,308	2,730,652	3,873,432

Table 4.3: Number of owl:sameAs links per pair of KBs.

4.5.2 Online Relation Alignment Setup: Sampling Strategies

One main *efficiency* aspect which allows us to carry the relation alignment process in an online setting are the sampling strategies. We evaluate the *efficiency* and respectively the *effectiveness* of our approach by computing features and generating relation alignment candidates from a set of sampled entity instances. A detailed description of the sampling strategies is provided in Section 4.4.

In order to measure the right amount of information we need from a source KB such that our models are optimal in terms of performance and efficiency, we consider a varying set of sample sizes, namely {50, 100, 200, 500, 1000}.

In our setting we have three sampling strategies, *first–N, random*, and *stratified*. In the case of stratified, where we need to compute the different strata for a given relation r_S from KB_S which relies on entity types that come from the domain of r_S.

Here, the entity types will correspond to types in the DBpedia type taxonomy [12]. We take advantage of the fact that all sampled entities from the various KBs have equivalent entities in DBpedia. We opt for the DBpedia taxonomy due to the fact that the types form a hierarchy, contrary to type taxonomies from other KBs, where the type structure forms a directed-acyclic-graph (DAG). This presents a pre-condition for the *stratified sampling* where the *strata* must consist of disjoint sets of entities, while such constraint cannot be enforced in DAGs where one cannot determine the highest depth in the type structure for an entity.

Since we sample entities that are explicitly connected through owl:sameAs statements, we can choose the type representation from any of the KBs by simply extracting statements for relation rdf:type from a desired KB. It must be noted that in the case of the KB pair $\langle YAGO, Freebase \rangle$ we extract the owl:sameAs links, by using the DBpedia entities as proxy which has owl:sameAs links to both KBs.

12. http://mappings.dbpedia.org/server/ontology/classes/

4.5.3 Ground-Truth Construction

We construct manually the ground-truth for the relation alignment process with the help of experts. The ground-truth is constructed for each KB pair, thus resulting in a total of 6 pairs. To guide the ground-truth construction process, for each relation pair $\langle r_S, r_T \rangle$ we compute *pca* and *cwa* features in Section 4.3.2 on the full set of entities. Giving those results as a bases to the experts is initialized the process of the ground-truth construction. However, this by no means represent the single rule that is used for annotation, in many cases, high *pca* and *cwa* scores have results on the relation alignment pair being irrelevant. Therefore, we manually check based on the relation names, the entities associated with such relation to come up with the judgments. In our ground-truth we keep only the pairs for which $r_S \Rightarrow r_T$ holds.

4.5.4 Evaluation Metrics

We evaluate our relation alignment model on two main aspects: (i) accuracy and (ii) efficiency. In terms of accuracy, we compute standard evaluation metrics like *precision, recall* and *F1* score. While, for *efficiency* we measure the introduced overhead in the query-execution process, which we measure in terms of *time* and *network bandwidth* usage. Below we describe in details the individual metrics.

– **Precision – P:** Is the ratio of *correctly* labeled relation alignments (that conform to the labels in our ground-truth) over the total of relation pairs predicted as *'correct'* by our model. More formally, precision is computed based on the formula:

$$P = \frac{true\ positives}{true\ positives + false\ positives} \tag{4.24}$$

– **Recall – R:** Is the ratio of the *correctly* labeled relation alignment by our classifier over the total of possible correct alignments in our ground-truth. More formally precision is computed based on the formula:

$$R = \frac{true\ positives}{true\ positives + false\ negatives} \tag{4.25}$$

– **F1 Score – F1:** It is the harmonic mean of precision and recall scores. Given by the formula:

$$F1 = 2 \times \frac{P \times R}{P + R} \tag{4.26}$$

– **Time – t:** Here we measure the amount of time taken to sample for S_{r_s}.
– **Network Bandwidth Usage – b:** It is measured as the total number of bytes it requires for the different sample sizes in S_{r_s}.

4.5.5 Learning Framework: Relation Alignment Models

Here we describe the learning framework for our relation alignment models. In Section 4.3.3 we described two supervised machine learning models, namely, *Voted Perceptron* (VP) and *Logistic Regression* (LR).

We are interested on evaluating and assessing the following aspects:

1. Which supervised model yields the optimal performance (in terms of P/R/F1) for the relation alignment problem?
2. What are the most influential feature groups for the relation alignment models?
3. Which is the best performing sampling strategy?
4. How much training data do we need in order for our alignment models to converge?
5. How does a relation alignment model generalize across different KB pairs?

In terms of (1) we assess the performance of the *VP* and *LR* for all KB pairs in our experimental setup. We evaluate the models on the full set of *relation alignment* candidates and consider a 5-fold cross-validation [57] [13]. This is done in order to assess the *real performance* of the models and determine the optimal model (either VP or LR).

For (2), we ran the *feature selection* algorithm as described in Section 4.3.3, and choose the corresponding top–k that provide optimal performance for the trained models, and the choice of top–k ranked features for the different KB pairs in our experimental evaluation.

In the case of (3) and (4), we show which sampling strategy has the best performance, and analyze the amount of training instances (relation alignment pairs) required for the models to converge. We analyse this aspect for the best model as addressed in (1).

Finally, in (5) we consider how well our models generalize across different KB pairs. We consider if a model learned on a specific KB pair will perform well on other KB pairs, hence, we analyze the implications of such aspect and the impact it has on the performance of our relation alignment models.

4.5.6 Baselines

Our approach is evaluated and compared to two baseline approaches on the domain of relation alignment using rule based techniques. The first one is a system called *PARIS*[102] implementing *cwa* measure and the second one is a system called *ROSA*[46] implementing *pca* measure.

We implemented independently *pca* and *cwa* and run the same experiments as to our approach. In order to be fair with the methods we run the experiments with their best configuration. We run an experiment to figure out which is the threshold where *pca* and *cwa* perform the best. Figure 4.3 presents the average F1 while in Figure 4.4 is shown the averaged precision and recall.

13. The k–fold cross–validation technique is done as following. The original set of results is randomly partitioned into k (5 in our case) equal size sets (folds). One of the k folds is used as the validation data to test the model and the remaining k-1 subsets are used to train the model. The cross-validation process is repeated k times, and the results from the k iterations are averaged produce the final estimation of the model.

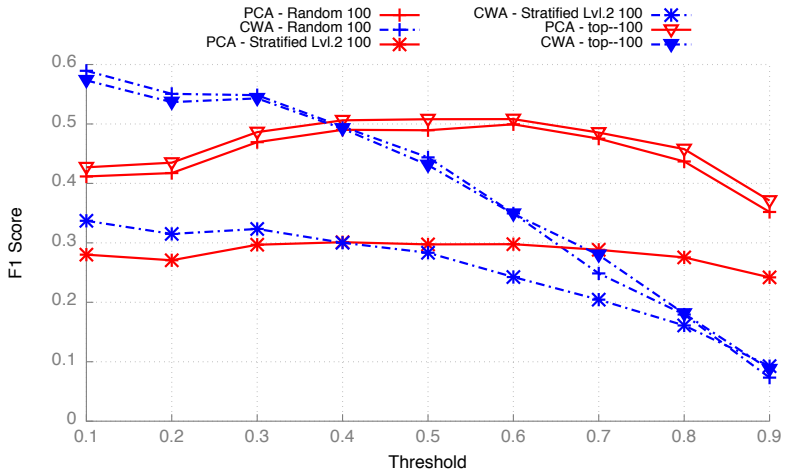
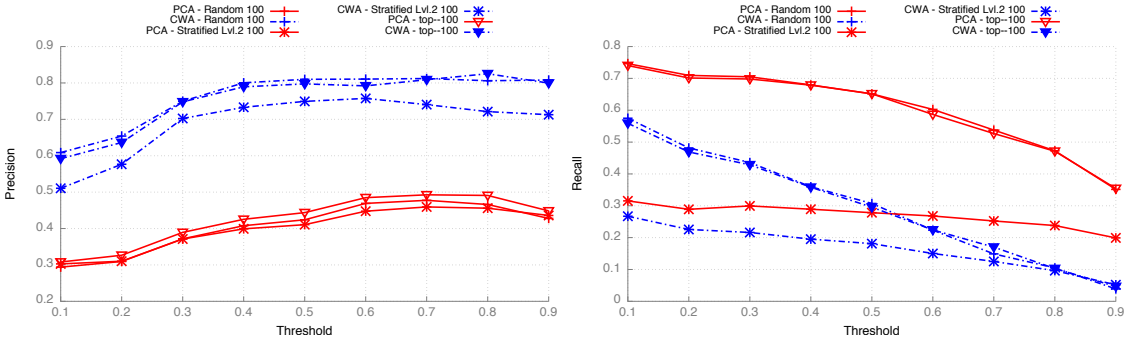


Figure 4.3: The average F1 score of the baselines *pca* and *cwa*. In x–axis we show the different threshold cut-offs (*pca* and *cwa* scores), which we use to determine whether a relation alignment pair (if its above a predetermined threshold) is *correct* or *incorrect*.



(a) avg. Precision

(b) avg. Recall

Figure 4.4: The average precision and recall scores for the baselines *pca* and *cwa*. In x–axis we show the different threshold cut-offs (*pca* and *cwa* scores), which we use to determine whether a relation alignment pair (if its above a predetermined threshold) is *correct* or *incorrect*.

4.6 Results and Discussion

In this section, we present the results of our experimental evaluation and discuss the implications of the individual factors performance and efficiency factors we encode in our online relation alignment model. Specifically, we address the main points we single out in our learning framework in Section 4.5.5.

4.6.1 Relation Alignment Model Performance

Here we evaluate the different supervised machine learning models, respectively their performance for the relation alignment problem. In more details, in this section, we ad-

dress the following questions.

- **Question–1:** Which supervised model yields the best performance for the relation alignment problem?
- **Question–2:** What are the most influential feature groups for the relation alignment models?

Best Model: Question–1. In Section 4.3.3 we introduce two different linear supervised models: *voted perceptron* (VP) and *logisitic regression* (LR). We train such models on the feature set in Table 4.1.

Table 4.4 shows a comparison of the results we achieve for VP and LR. We train these models on the full set of entity instances for all KB pairs under comparison. Furthermore, we learn the models based on the complete set of features and assess their performance with 5 fold cross-validation. The aim here is to single out the best performing model.

It is evident from Table 4.4 that the results computed based on the LR model are significantly better when compared to VP for all KB pairs. The same holds when compared against our baselines.

In Table 4.4 we show as well the results of our baselines *pca* and *cwa* presented in Section 4.5.6. We apply these two measure are already used in ROSA [46] and PARIS [102] systems for relation alignments, which are produced using the full set of entities. To ensure a fair comparison, for both measures we use their best configuration, namely the thresholds we extract for the averaged F1 score as shown in Figure 4.3. For *pca* we consider the relation alignments with *pca* score above 0.3, whereas for *cwa* we consider the relation alignments with *cwa* score above 0.1.

		LR			cwa 0.1			pca 0.3			VP		
KB_S	KB_T	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Y	D	0.92	0.73	**0.81**	0.27	0.48	0.35	0.06	0.56	0.11	0.83	0.33	0.48
D	Y	0.57	0.49	**0.53**	0.33	0.34	0.34	0.18	0.33	0.24	0.36	0.26	0.30
Y	F	0.82	0.82	**0.82**	0.40	1.00	0.57	0.03	1.00	0.05	0.73	0.73	0.73
D	F	0.69	0.38	**0.49**	0.31	0.65	0.42	0.05	0.85	0.09	0.05	0.03	0.04
F	Y	0.69	0.74	**0.71**	0.73	0.60	0.66	0.61	0.86	0.71	0.56	0.44	0.49
F	D	0.87	0.66	**0.75**	0.72	0.57	0.64	0.34	0.93	0.50	0.60	0.50	0.54
average		**0.76**	**0.64**	**0.69**	0.46	0.61	0.49	0.21	0.75	0.28	0.52	0.38	0.43

Table 4.4: Comparison of *Logistic Regression* (LR) and *Voted Perceptron* (VP) models for the relation alignment problem. Comparison with the baselines *cwa* and *pca* on the full dataset. [a]

a. Please note that we replaced the full names of the knowledge bases for spacing reasons. Y = YAGO, D = DBpedia, and F = Freebase.

Feature Selection: Question–2. One precaution we need to take into consideration is *over-fitting*. It is evident that the higher the number of features our supervised models will perform better on the training instances, however, this poses a risk of over-fitting if the number of features is too high, and thus failing to generalize over unseen instances. Hence, we consider a standard mechanism for supervised learning, namely *feature selection* in Section 4.3.3.

In this question we consider which features are most influential for the problem of relation alignment. Namely, we consider the impact of individual features on the performance of our models, and the performance of our models when we use the top–5 performing features. We take only top–5 features as this represents roughly 50% of the computed features in Table 4.1.

In Table 4.5 we show the results similar to those in Table 4.4, however, by training the supervised models on the top–5 features. We select the top features through the introduced *feature selection* algorithm in Section 4.3.3. The algorithm relies on the *information gain* each individual feature provides to the model on classifying correctly the relation pairs $\langle r_S, r_T \rangle$, consequentially ranks them according to that score.

It is evident that by considering top–5 features for learning our models, the results are highly similar. In terms of $F1$ score, we achieve the same results. The advantages from such results are two-fold: (i) we reduce the complexity of our models, hence, generalizing well on unseen relation pairs, and (ii) we do not lose in terms of performance.

		LR			VP		
KB_S	KB_T	P	R	F1	P	R	F1
YAGO	DBpedia	0.90	0.66	0.76	0.87	0.46	0.60
DBpedia	YAGO	0.56	0.45	0.50	0.3	0.34	0.32
YAGO	Freebase	0.81	0.81	0.81	0.66	0.72	0.69
DBpedia	Freebase	0.58	0.32	0.41	0.014	0.009	0.01
Freebase	YAGO	0.70	0.59	0.64	0.69	0.68	0.68
Freebase	DBpedia	0.79	0.55	0.65	0.50	0.366	0.42
average		0,72	0,56	0,63	0,50	0,43	0,45

Table 4.5: Comparison of *Linear Regression* (LR) and *Voted Perceptron (VP) models for the relation alignment problem learned with only the top–5 features selected through the feature selection algorithm based on* Information Gain*. We evaluate the models using 5-fold cross-validation for all KB pairs.*

Feature Ablation. An important aspect we consider in our experimental setup is the impact of the different feature groups for the relation alignment models. We show the impact of the individual features for the models trained based on the *LR* model. We evaluate the models based on the 5 fold cross-validation approach, and by computing the feature values based on the full set of entity instances for a relation r_S.

Figure 4.5 shows the feature ablation results for the different feature groups in Table 4.1 for the relation alignment problem.

We note that, the highest impact is attributed to the *GRS* feature group. This follows our intuition where we hypothesized that for a relation to be subsumed in a target relation, one of the important factors is to be able to represent the entity instances, namely the entity types for a given relation should be similar. Furthermore, the next assumption we made was that *ILP* feature scores, depending on the KB pairs, sometimes are insufficient (see Section 4.3.2), hence, the corresponding likelihood scores of the ILP measures can provide additional information on predicting correctly the label of a relation pair.

In the next group, that of ILP features, we note that they have a significantly lower performance as measured through our performance metrics. We acknowledge here that some of the features are correlated (e.g. pca and $p(correct|pca)$), however, in the previous group the respective scores are contextualized in terms of likelihood measures for the specific KB pairs.

Finally, we notice that some of the features like the *lexical similarity* may not provide much additional gain in terms of performance. That is, due to the naming conventions and names of the relations in the different KB pairs. However, this presents a general purpose feature which will provide additional performance gain in cases where the KB pairs under consideration will have some form of similarity in the naming convention of the relations.

In summary, we see that the combination of all the features yields the best performance of our model on the full datasets, with the *GRS* feature group providing the most contribution in terms of performance, whereas the remainder of the features contributes to an improvement of up to 5%.

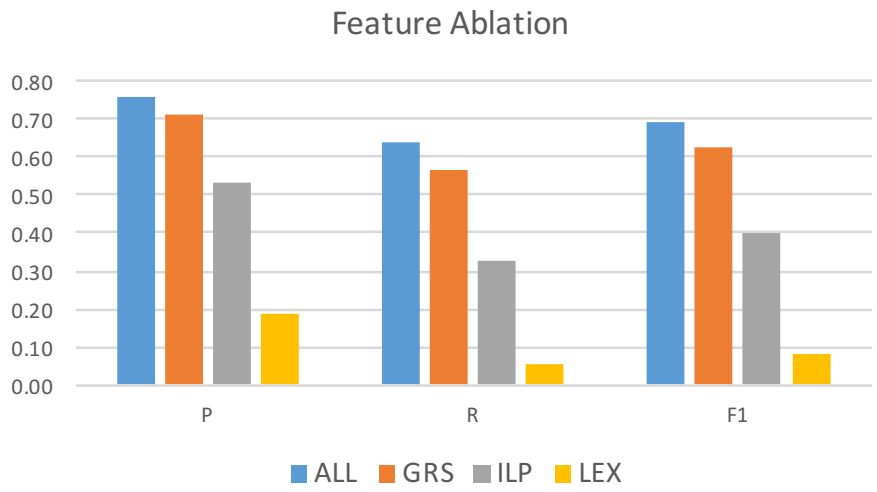


Figure 4.5: Feature ablation for the relation alignment model trained with *logistic regression*. The scores for precision/recall and F1 are averaged across the different KB pairs. The results are shown for the different feature groups.

4.6.2 Efficient Relation Alignment

One main challenge that we address in this Chapter deals with the *efficiency* of our approach. Efficiency aspects present one of the main drawbacks of related work in ontology matching, given the scale and the large number of datasets.

In this section we answer the following questions that deal with the efficiency of our approach. It is worth noting that the relation alignment models here correspond to those learned through the LR approach and using the top–5 features.

- **Question 3:** Which is the best performing sampling strategy?
- **Question 4:** How much training data do we need in order for our alignment models to converge?

The first precaution we take into account for the efficiency of our approach, is through sampling of entity instances that have as a relation r_S from our relation alignment pair $\langle r_S, r_T \rangle$. We introduced three sampling strategies in Section 4.4.

Sampling Configurations: Question–3. Here we show which of the sampling strategies achieves the best results for the relation alignment problem. Here we show the results only for the best performing model trained based on *LR* model.

The features in this case are computed based on samples of entity instances (contrary to the previous section where we compute the features based on the full set of entity instances for r_S). As parameters in Table 4.6 we vary the number of entity samples (for $\{50, 100, 500, 1000\}$), and the strategies *first–N, random* and *stratified* (with different configurations on how we construct the *strata*).

Through sampling we ensure that we require minimal access to the respective KBs for which we want to align the relations. From Table 4.6 we see that the best configuration is the *stratified sampling* strategy with *strata* constructed with entity types from the DBpedia taxonomy up to level 3 [14]. Furthermore, the best results are achieved with a minimal sample of entity instances, namely, 50 entity instances that are connected to r_S.

In addition, we train and evaluate the models for the different sampling configurations. Since at this point we are interested on only finding the best performing sampling strategy, we vary the amount of training instances (5%, 10%, ..., 50%) we use to learn our models, with up to a maximum of 50% instances for training. Here the results are averaged across the different KB pairs and for the varying percentage of instances used for training our models.

We do this, in order to find the best configuration with the following parameters: (i) sampling strategy, and (ii) entity samples.

Furthermore, it is interesting to note that we do not achieve significant improvement with the increase of entity samples. With already 50 entity samples, the models for the different sampling strategies seem to converge. However, we note an insignificant oscillation of results across the different entity sample sizes.

14. Strata computed at level 3 represents entity types from DBpedia taxonomy with a depth of at most 3 from the root of the type taxonomy.

Sampling	50			100			500			1000		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
firstN	0.77	0.61	0.67	**0.78**	**0.59**	**0.66**	0.80	0.54	0.63	0.73	0.51	0.59
random	0.78	0.62	0.68	**0.82**	**0.55**	**0.65**	0.77	0.45	0.55	0.77	0.50	0.59
str.lvl–2	0.72	0.58	0.62	0.81	0.54	0.63	0.76	0.56	0.63	0.77	0.54	0.62
str.lvl–3	**0.81**	**0.59**	**0.67**	0.78	0.59	0.65	0.76	0.49	0.59	0.75	0.53	0.62
str.lvl–4	0.74	0.53	0.60	0.76	0.52	0.60	0.77	0.55	0.62	0.74	0.53	0.59
str.lvl–5	0.75	0.51	0.60	0.76	0.55	0.63	0.82	0.53	0.63	0.77	0.49	0.57
str.lvl–6	0.72	0.52	0.58	0.72	0.55	0.62	0.82	0.59	0.66	0.80	0.59	0.67

Table 4.6: The performance of the relation model trained using the *logistic regression* model for the different sampling strategies and entity sample instances. We average the results across all KB pairs and for the varying amount of training data we use to train our models, we limit here the training amount of information up to 50%. We select the best configuration (marked in bold) from the above sampling strategies and sampled instances taking into account the highest P and where the R score is reasonably high.

Model Convergence: Question–4. Here, we show the amount of training instances required for the models to converge and achieve the best performance. Table 4.7 shows the best relation alignment models, and the *convergence* of the models with the increasing amount of training instances used for learning the models. We see that our best relation alignment model based on *stratified sampling* strategy, achieves the best performance with only 10% of training instances. In terms of precision, the best results are achieved by using 30% of instances for training.

train %	firstN-100			random-100			str.lvl-3-50		
	P	R	F1	P	R	F1	P	R	F1
5%	0.69	0.59	0.60	0.69	0.59	0.60	0.73	0.63	0.66
10%	0.77	0.60	0.64	0.77	0.60	0.64	0.81	0.62	0.70
20%	0.79	0.61	0.67	0.79	0.61	0.67	0.85	0.58	0.68
30%	0.83	0.55	0.66	0.83	0.55	0.66	**0.85**	**0.59**	**0.69**
40%	0.83	0.60	0.69	0.83	0.60	0.69	0.82	0.56	0.66
50%	0.80	0.62	0.69	0.80	0.62	0.69	0.78	0.56	0.64

Table 4.7: Comparison of the best relation alignment models under the different sampling strategies with varying percentage of training instances from the full set of relation alignment candidates. The results are average across all KB pairs for comparison purposes.

Finally, from Table 4.7 we found out that the best model and configuration to use for the relation alignment problem, is when we use the *stratified sampling* with 50 samples of entity instances, and learning the *LR* model with 30% of instances for training (one could opt for the highest $F1$ scores, hence, use only 10% for training).

The corresponding results for the individual KB pairs are shown in Table 4.8. We see that, we have significantly higher performance for most of the KB pairs when compared to our baselines. For some cases like $\langle YAGO, DBpedia\rangle$, $\langle Freebase, DBpedia\rangle$, etc., we have better performance scores for both precision and recall. In the case of baselines, we note that as argued in Section 4.3, the models based on pca allow for too many false positives, and due to its relaxed constraints regarding relation subsumption. In the case of cwa the recall scores are worse than pca.

		LR			pca			cwa		
KB_S	KB_T	P	R	F1	P	R	F1	P	R	F1
DBpedia	Freebase	0.79	0.33	0.47	0.10	0.67	0.18	0.31	0.5	0.40
DBpedia	YAGO	0.87	0.70	0.77	0.30	0.72	0.43	0.70	0.66	0.68
Freebase	DBpedia	0.93	0.53	0.68	0.27	0.79	0.41	0.65	0.65	0.65
Freebase	YAGO	0.70	0.58	0.64	0.22	0.39	0.28	0.42	0.37	0.39
YAGO	DBpedia	1.00	0.66	0.79	0.17	0.75	0.28	0.71	0.66	0.68
YAGO	Freebase	0.83	0.77	0.80	0.11	0.78	0.20	0.55	0.59	0.57

Table 4.8: The results for the individual KB pairs computed based on the *LR* model for stratified sampling (level–3 and with 50 sampled entity instances) when using 30% for training and the rest for evaluating the performance of the models.

Noteworthy is the comparison of the average performance scores for the best performing model in Table 4.8 and the results we achieve in Table 4.4. In terms of F1 the results are the same, and with better recall in the case of the models trained on the full data. However, in terms of precision we note a difference of 6%, we believe such difference might be an artifact of the followed evaluation strategies. Since in the first case we are interested in finding the best supervised machine learning model, and thus are not interested in optimizing for efficiency, we considered the k-fold cross validation. Contrary to the second case where our aim is efficiency (in terms of training data, sampling strategy etc.) and effectiveness of our approaches, hence, we evaluate the models by splitting them into train/test instances.

4.6.3 Generalizing Relation Alignment Models

In the previous sections, we presented results based on models that were computed for specific KB pairs. It is evident from the datasets descriptions in Section 4.5.1 that they are inherently different, in terms of the number of relations and entities they contain. Here we answer the question on how well a model trained on a specific KB pair will generalize on other KB pairs. That is, if we train on $\langle KB_S, KB_T\rangle$ for a specific number of training instances, how well do the models perform on assessing the relevance of the relation alignment pairs for *other* KB pairs.

– **Question–5.** How do relation alignment models generalize across KB pairs?

Model Generalization: Question–5. Table 4.9 shows the results we achieve when training on the KB pairs, and evaluating on the remaining KB pairs (inclusive of the one KB pair we use for training). We use a fairly small amount of relation alignment pair instances, namely from 50 up to 200 for training. The results show the average precision across all KB pairs used for the evaluation.

The results show that our model generalize well and do not overfit if trained on a specific KB pair. Furthermore, we see that the drop in terms of performance is not significant. That is, if we take the KB pair $\langle DBpedia, Freebase \rangle$, with a small amount of training instances we perform reasonably well across the remaining KB pairs.

In conclusion, this shows that the decision we undertake towards constructing the relation alignment models, such as the choice of features, the feature selection, and lastly the efficiency approaches through sampling, we can compute with high accuracy the relation subsumption across KBs.

KB_S	KB_T	#Instances			$Avg(P)$
		50	100	200	
DBpedia	Freebase	0.86	0.82	0.96	0.88
DBpedia	YAGO	0.77	0.85	0.80	0.81
Freebase	DBpedia	0.25	0.29	0.83	0.46
Freebase	YAGO	0.25	0.85	0.79	0.63
YAGO	DBpedia	0.73	0.67	0.79	0.73
YAGO	Freebase	0.70	0.75	0.75	0.73

Table 4.9: The models correspond to the best configuration (stratified–level-3 with 50 sampled entity instances) which we train on one KB pair, $\langle KB_S, KB_T \rangle$, with a specific number of training instances, and evaluate on the remaining KB pairs. The results show the $Avg(P)$ score across all KB pairs, and emphasize how well these models generalize across KB pairs.

It is interesting to note that the models that are trained with Freebase as the source knowledge base fail to generalize over other KB pairs. We hypothesize that this poor generalization might be attributed to the fact that the nature of relations in Freebase is significantly different from other KBs. In Freebase, relations are sparse, meaning they have a low number of statements and thus the computed features scores will differ when computed for other pairs. Hence, the models trained on those scores will not perform well in other pairs. However, for models that are trained on other KB pairs like DBpedia or YAGO, they generalize well across all KB pairs.

4.6.4 Coverage

Table 4.10 shows the portion of r_{ee} relations from a source KB_S that are subsumed in a target relation in KB_T. We denote this portion as the *coverage* of r_{ee} relations from KB_S in KB_T. We measure the *coverage* based on the constructed ground-truth (see Section 4.5.3), which consists of relations that are aligned by our expert annotators (aligned relations

from a source KB_S to a target knowledge base KB_T). The *coverage* is the ratio of the number of distinct aligned relations r_S over the total distinct number of relations in KB_S. The total number of relations in KB_S, and the coverage of relations from KB_S into KB_T is shown in the third and fourth column in Table 4.10.

The main aim of the *coverage* measure is to show the impact on aligning knowledge bases, respectively their r_{ee} relations. It is evident from Table 4.10 that the coverage varies across the different KB pairs. We attribute such variation in coverage mainly to two factors: (i) quality of a KB, and (ii) the granularity of relations in a KB.

For (i), we notice that in the case of Freebase the number of relations is significantly higher in comparison to YAGO and DBpedia. The reason for this is that Freebase is a collaboratively created KB, including the relations and the corresponding facts for such relations. This leads to a large number of relations, where different relations describe similar concepts. This has several implications, where some of the relations will have a low number of facts associated and in some cases these facts are erroneous, and therefore, have low overlapping facts with other KBs.

In the case of (ii), we notice that depending on the target KB, the *coverage* will vary. For instance, if our target knowledge base is Freebase, we see that the coverage is low. This is due to the fact that, the relations in Freebase are fine grained and as such do not subsume more coarse grained relations coming from YAGO or DBpedia. It becomes clearer when DBpedia is the target KB, in which case the *coverage* is significantly higher.

Finally, in the fifth column in Table 4.10 we show the coverage of our approach on aligning subsumed relations for a given KB pair. The *coverage* is measured relative to the coverage of aligned relations based on our ground-truth. We note that in majority of the cases we perform reasonably well with an average coverage of 0.78. This is inline with the recall scores we achieve in Table 4.4. However, in the case of *recall* we measured w.r.t all possible target relations for a given source relation, whereas in the case of *coverage*, if a source relation is aligned to *at least* one target relations then we mark it as covered in the target knowledge base KB_T.

KB_S	KB_T	r_{ee} in KB_S	r_{ee} - aligned between KBs	r_{ee} - aligned by SOFYA
YAGO	DBpedia	36	0.75	1
YAGO	Freebase	36	0.33	1
DBpedia	YAGO	563	0.23	0.78
DBpedia	Freebase	563	0.19	0.79
Freebase	YAGO	1666	0.08	0.5
Freebase	DBpedia	1666	0.44	0.61

Table 4.10: Coverage of the relations for each KB pair.

4.6.5 Query-Execution Overhead

Here, we evaluate one important aspect of our relation alignment model, namely, the efficiency aspects of our approach. These represent important features considering that

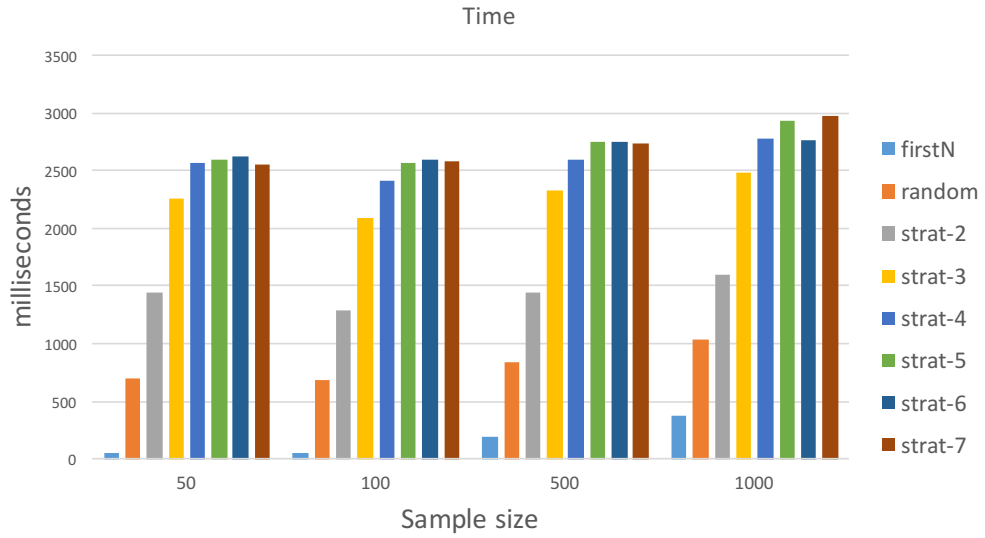


Figure 4.6: Time statistics in milliseconds for the different sampling strategies of Section 4.4, for different sample sizes. Averaged through all the KB pairs.

our relation alignment is setup in an online setting. We evaluate for two main efficiency metrics as introduced in our evaluation metrics in Section 4.5.4, we measure the overhead in terms of *time* and *network bandwidth usage*.

In our approach we introduce an overhead on the query-execution (taking into account that our relation alignment is performed in an online setting), namely, in the *entity instance sampling* process and the amount of extra bandwidth we use by transferring the entity samples through the network.

Query-Execution: Time overhead. With respect to *time* efficiency factor, Figure 4.6 shows the amount of time (in milliseconds) it takes to perform the different sampling strategies, and for the varying amount of entity samples. It is evident that the most efficient sampling strategy is *first–N* taking the least amount of time. This is intuitive as we select the first matching statements for relation r_S. In contrast to the other approaches, we see a significant increase on the amount of time it takes to produce the desired set of samples. For instance, the *stratified* sampling strategy requires the most amount of time, with a maximum of 3 *seconds*. This in comparison to other sampling strategies is significantly higher, however, considering response time of SPARQL endpoints, such an overhead does not represent a huge overhead on their response time.

Query-Execution: Network bandwidth overhead. In Figure 4.7 we present the second efficiency factor, namely the amount of overhead introduced in terms of *bandwidth*. Similarly as for *time* overhead, here too, we compare all sampling strategies. Understandably the amount of overhead in terms of bandwidth is uniformly distributed across the different sampling strategies. That is, considering that we sample for the same amount of entity instances. The highest amount of bandwidth overhead is introduced when we

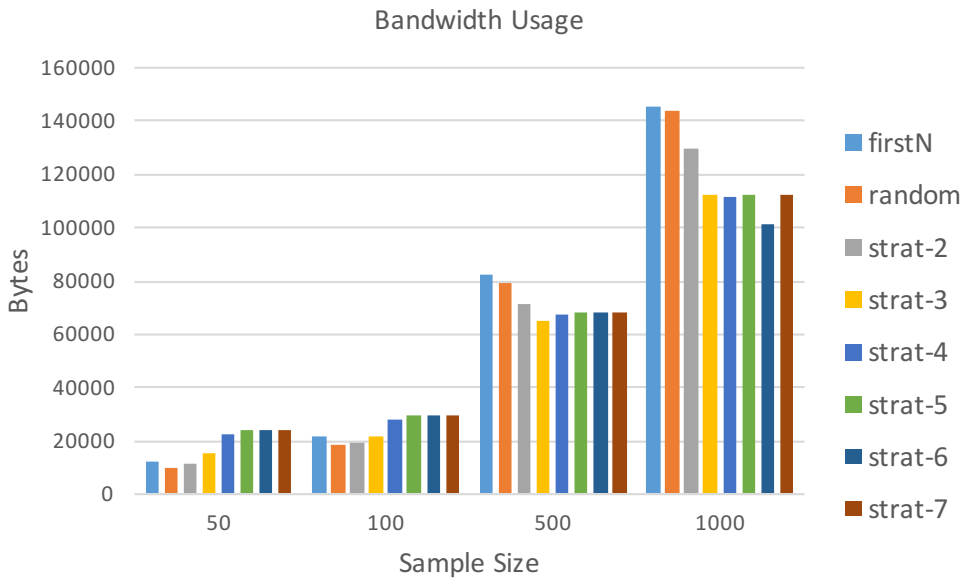


Figure 4.7: Bandwidth usage statistics in bytes for the different sampling strategies of Section 4.4, for different sample sizes. Averaged through all the KB pairs.

sample for 1000 entity instances for r_S, with a maximum 140 *kb*.

However, in the performance evaluation of our relation alignment model in Table 4.7, we found that the best results are achieved with entity sample instances ranging between 50 and 100, for *stratified* and *first–N, random*, respectively. In terms of bandwidth overhead, this results into a maximum of 20 *kb*, which represents a reasonably low overhead in terms of bandwidth.

4.7 Conclusion

In this chapter, we presented a relation alignment approach for knowledge bases. For a given source relation r_S from a source knowledge base KB_S, we find candidate relations r_T in a target knowledge base KB_T, and assess whether for $\langle r_S, r_T \rangle$ it holds that r_S is subsumed by the target relation r_T, that is, $r_S \Rightarrow r_T$.

For the relation alignment process we employ supervised machine learning models, namely *Voted Perceptron* and *Logistic Regression*, which we learn on a set of features that make use *ILP* measures, namely pca and cwa for the pair $\langle r_S, r_T \rangle$. In addition, we consider *general relation statistics* that exploit the landscape of the relations in terms of entity type distributions and the corresponding relation name similarity between a relation pair. Finally, given that for different KB pairs under consideration, the feature values may vary, hence, we additionally consider the *likelihood* of specific ILP feature values predicting the relevance of relation pair alignment.

Furthermore, we address one of the main drawbacks of existing work in *ontology matching*, namely the efficiency of such approaches. We perform the relation alignment

process in an *online setting*, where for r_S we sample a minimal set of entity instances from KB_S which we use to discover candidate relations for assignment in KB_T. For this purpose we employ three different sampling strategies, namely, *first–N, random* and *stratified*. The sampling process is geared towards improving the *coverage* of target relations in KB_T while maintaining the efficiency of our alignment models.

Finally, we perform an extensive evaluation on three real-world knowledge bases, DBpedia, YAGO, and Freebase. We test the following factors in our experimental evaluation.

1. **Performance of Supervised Models.** We find out that the best performance is achieved through the *logistic regression*, which proves to be better in our case due to its optimal weighting scheme of the features.
2. **Feature Impact.** We find out that the highest impact is achieved through the GRS feature group.
3. **Optimal Sampling Strategies.** In terms of sampling, we find out that *stratified sampling* with *strata* constructed at the depth level 3 in the DBpedia type taxonomy provides the optimal coverage and performance.
4. **Convergence of Relation Alignment Models.** The models which we train with the best configuration (logistic regression with stratified sampling) we find out that they converge with as nearly as 10% used for training from the relation pair candidates.
5. **Generalization of Relation Alignment Models.** We assess how well our models generalize across the KB pairs. That is, we train a model by taking only 50 or 100 relation pairs as training instances from a specific KB pair and evaluate on all the other KB pairs.
6. **Query-Execution Overhead.** The last factor we assess is the introduced overhead in terms of query-execution for our relation alignment model. The amount of time required to perform the sampling is highest for *stratified sampling*, while the bandwidth overhead is uniform across the different sampling strategies. We conclude that at worst we introduce 3 seconds, which for the existing Linked Open Data infrastructure does not present a high delay in query execution.

Chapter 5

Conclusion and Perspectives

In this thesis, we identified several challenges that deal with the integration of Web services and linked datasets. The challenges we address deal with the mapping of output from Web service operation calls into an RDF global schema, in our case coming from real-world knowledge bases like YAGO or DBpedia. Next, we address the problem of ontological relation alignment in linked datasets. We performed such process in an online setting, addressing some of the shortcoming of related work that deal with schema alignment at class level, namely, the full access to datasets, and relation alignment (a problem largely untouched so far by related work).

The outcomes and contributions of the approaches proposed in *DORIS* and *SOFYA* are the following. Through the mapping of output to global schemas, we allow for seamless integration of services into mashups, where the output is interpreted uniformly through the schema. Whereas, with the alignment of relations, we allow users access to multiple datasets for a given relation through *query rewriting*, hence, allowing for a more complete view of existing information from the large number of linked datasets.

Finally, the approaches in *DORIS* and *SOFYA*, leads towards a model for uniformly accessing and integrating data coming from different datasets with heterogeneous structures, i.e. RDF datasets or Web service APIs.

In the following, we summarize the contributions in the individual chapters and then discuss about future lines of work.

5.1 Thesis Summary

Mapping of Web service output into a global RDF schema. We presented *DORIS*, a method that automatically generates a mapping of REST Web service output into the terms of a given global schema, with the following contributions:

- An algorithm that formally describes the output of a given Web service in terms of a global schema, as a view with binding patterns over that schema.
- Exploit instances from real-world knowledge bases to obtain output from a specific operation from a Web service. Finally, exploit the intersection between instances from a KB and the output from an operation call for the mapping process.

- Transformation functions as XSLT scripts, which transform the output of a Web service in terms of a global schema.
- Approaches to discover Input/Output dependencies between Web services within the same API, particularly useful for Web services that use identifiers for input.
- Extensive experimental evaluation on real-world Web services from four different domains and on three real-world RDF knowledge bases (DBpedia, Yago and BNF).

Online Relation Alignment for Linked Datasets. We presented *SOFYA*, an ontological relation alignment approach for knowledge bases, for which we make the following contributions:

- An instance-base relation alignment approach that discovers subsumpsion relationships for a relation r_S from a source knowledge base to a relation r_T in a target knowledge base.
- Supervised machine learning models which combine features that exploit *association rule mining* measures, *general statistics* about the relation pair, and lexical similarity of relation names, to label as either *correct* or *incorrect* the subsumption alignment between a relation pair $\langle r_S, r_T \rangle$.
- An efficient relation alignment approach, which uses a small sample of the data to compute subsumption alignments between relations.
- Extensive experimental evaluation on three real-world RDF knowledge bases (DBpedia, YAGO and Freebase).

5.2 Future Work

In this section, we outline possible avenues for future work. We start by giving short-term plans for future works and continue with more long-term plans.

Automatic discovery of the input type in DORIS. In the current implementation of our approach *DORIS*, we assume that the type of the input of a Web service is given. Especially, in the case when we first deal with an API we did not align before in order to apply the approach for *discovering I/O dependencies*.

The plan for future work is to provide an automatic or semi-automatic way to discover the input type of Web services. A possible direction is to use knowledge bases and continuously probe the Web service with entities from different *classes* until we find the one that leads to meaningful call results.

Extend the experimental evaluation in DORIS. To this level, we evaluated our DORIS system using Web services that expect only one value as input. As future work we plan to extend the evaluation of our system with Web services that use more than one input value.

Entity-literal relations. For the moment the approach in *SOFYA* considers only entity-entity relations (r_{ee}) between knowledge bases. As future work we aim at extending it further and consider entity-literal relations (r_{el}). This adds an extra challenge since in the r_{el} relations the `owl:sameAs` relation holds only for the subject entities, since objects are literals. A possible direction to overcome such an obstacle is to apply string similarity techniques and compare the literals between the relations.

Additional Features One of the task that we would like to investigate more is what other possible features (*matchers*) could be used to improve the already sufficiently well-performing relation alignment models. As for the already existing features we already mentioned in Section 4.3.2, that a future direction is to further enrich lexical similarity features with external dictionaries or thesaurus to match the relation names.

Relation alignment for complex relations. At this stage of our work, we compute $1-1$ relation submsumption alignments. That is, we consider one relation from the source knowledge base and we search for the relation in which it is subsumed in the target knowledge base. For future work, we aim at extending our approach in order to be able handle subsumption relationships between compositions of $m-n$ relations. For instance, subsumption alignments with the composition of two relations may take the following form:

$$r_{S1}(x,y) \wedge r_{S2}(y,z) \Rightarrow r_T(x,z)$$

$$r_S(x,z) \Rightarrow r_{T1}(x,y) \wedge r_{T2}(y,z)$$

Use relation alignment for query rewriting. Another future perspective is to be able to compute at query time subsumptions of type: $r_S \Leftarrow r_T$. In this way for a given SPARQL conjunctive query Q_S over a given KB_S, and another KB_T, we will be able to find a rewriting Q_T of Q_S over KB_T at query time. In this rewriting, each atom relation of Q_S is rewritten in a atom relation in Q_T.

Bibliography

[1] Datasets by topical domain. http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/.

[2] https://developers.google.com/freebase/. https://developers.google.com/freebase/.

[3] The linked open data cloud. http://lod-cloud.net.

[4] Wikidata a collaboratively edited knowledge base. https://www.wikidata.org/wiki/.

[5] Wordnet lexical database for the english language. http://wordnetweb.princeton.edu.

[6] Alexa. http://www.alexa.com/topsites, 2016.

[7] Internet users in the world. http://www.internetlivestats.com/internet-users/, 2016.

[8] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[9] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas. The web changes everything: understanding the dynamics of web content. In R. A. Baeza-Yates, P. Boldi, B. A. Ribeiro-Neto, and B. B. Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 282–291. ACM, 2009.

[10] R. Alarcón, R. Saffie, N. Bravo, and J. Cabello. REST web service description for graph-based service discovery. In P. Cimiano, F. Frasincar, G. Houben, and D. Schwabe, editors, *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, volume 9114 of *Lecture Notes in Computer Science*, pages 461–478. Springer, 2015.

[11] S. Albagli, R. Ben-Eliyahu-Zohary, and S. E. Shimony. Markov network based ontology matching. *J. Comput. Syst. Sci.*, 78(1):105–118, 2012.

[12] P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors. *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Morgan Kaufmann, 2001.

[13] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, 2007.

[14] D. Aum"uller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *Proc. ACM SIGMOD Conference*, 2005.

[15] V. Bárány, M. Benedikt, and P. Bourhis. Access patterns and integrity constraints revisited. In W. Tan, G. Guerrini, B. Catania, and A. Gounaris, editors, *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 213–224. ACM, 2013.

[16] M. Benedikt, B. ten Cate, and E. Tsamoura. Generating low-cost plans from proofs. In R. Hull and M. Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 200–211. ACM, 2014.

[17] T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[18] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *PVLDB*, 4(11):695–701, 2011.

[19] C. M. Bishop. *Pattern recognition and machine learning*, volume 1. springer, 2006.

[20] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227, 2009.

[21] R. Blanco, B. B. Cambazoglu, P. Mika, and N. Torzec. Entity recommendations in web search. In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II*, volume 8219 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2013.

[22] A. Bordes and E. Gabrilovich. Constructing and mining web-scale knowledge graphs: KDD 2014 tutorial. In Macskassy et al. [72], page 1967.

[23] A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, 2009.

[24] J. Cardoso. Discovering semantic web services with and without a common ontology commitment. In *SCW*, pages 183–190. IEEE Computer Society, 2006.

[25] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[26] S. Castano, V. D. Antonellis, and S. D. C. di Vimercati. Global viewing of heterogeneous data sources. *IEEE Trans. Knowl. Data Eng.*, 13(2):277–297, 2001.

[27] V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2015.

[28] W. G. Cochran. *Sampling Techniques*. John Wiley, 1953.

[29] I. F. Cruz, H. Xiao, and F. Hsu. Peer-to-peer semantic integration of xml and rdf data sources. In G. Moro, S. Bergamaschi, and K. Aberer, editors, *AP2PC*, volume 3601 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2004.

[30] M. d'Aquin, A. Adamou, and S. Dietze. Assessing the educational linked data landscape. In *Web Science 2013 (co-located with ECRC), WebSci '13, Paris, France, May 2-4, 2013*, pages 43–46, 2013.

[31] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[32] C. Delobel, C. Reynaud, M. Rousset, J. Sirot, and D. Vodislav. Semantic integration in xyleme: a uniform tree-based approach. *Data Knowl. Eng.*, 44(3):267–298, 2003.

[33] N. Derouiche, B. Cautis, and T. Abdessalem. Automatic extraction of structured web data with domain knowledge. In A. Kementsietsidis and M. A. V. Salles, editors, *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 726–737. IEEE Computer Society, 2012.

[34] H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[35] A. Doan, P. M. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In S. Mehrotra and T. K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 509–520. ACM, 2001.

[36] A. Doan, P. M. Domingos, and A. Y. Levy. Learning source description for data integration. In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[37] A. Doan, J. Madhavan, P. M. Domingos, and A. Y. Halevy. Ontology matching: A machine learning approach. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 385–404. Springer, 2004.

[38] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In Macskassy et al. [72], pages 601–610.

[39] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[40] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services, 2004.

[41] A. DuVander. 7,000 apis: Twice as many as this time last year. `http://www.programmableweb.com/news/`

7000-apis-twice-many-time-last-year/2012/08/23, 2012.

[42] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In Hencsey et al. [53], pages 669–678.

[43] Y. Freund and R. E. Shapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[44] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, C. Schallhart, and C. Wang. DIADEM: thousands of websites to a single database. *PVLDB*, 7(14):1845–1856, 2014.

[45] L. Galárraga, G. Heitz, K. Murphy, and F. M. Suchanek. Canonicalizing open knowledge bases. In J. Li, X. S. Wang, M. N. Garofalakis, I. Soboroff, T. Suel, and M. Wang, editors, *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1679–1688. ACM, 2014.

[46] L. A. Galárraga, N. Preda, and F. M. Suchanek. Mining rules to align knowledge bases. In F. M. Suchanek, S. Riedel, S. Singh, and P. P. Talukdar, editors, *AKBC@CIKM*, pages 43–48. ACM, 2013.

[47] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, and S. B. Moon, editors, *WWW*, pages 413–422. International World Wide Web Conferences Steering Committee / ACM, 2013.

[48] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*, pages 436–445, Athens, Greece, 1997.

[49] A. Halevy. Answering queries using views - a survey. *The VLDB Journal*, 10(4):270–294, 2001.

[50] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[52] B. He and K. C. Chang. Statistical schema matching across web query interfaces. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 217–228. ACM, 2003.

[53] G. Hencsey, B. White, Y. R. Chen, L. Kovács, and S. Lawrence, editors. *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*. ACM, 2003.

[54] F. Janssen, F. Fallahi, J. Noessner, and H. Paulheim. Towards rule learning approaches to instance-based ontology matching. In J. Völker, H. Paulheim,

J. Lehmann, and M. Niepert, editors, *Proceedings of the First International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data, Heraklion, Greece, May 27, 2012*, volume 868 of *CEUR Workshop Proceedings*, pages 13–18. CEUR-WS.org, 2012.

[55] T. Kirsten, A. Thor, and E. Rahm. Instance-based matching of large life science ontologies. In *DILS Workshop*, 2007.

[56] M. Klusch, B. Fries, and K. P. Sycara. OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services. *J. Web Sem.*, 7(2):121–133, 2009.

[57] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, pages 1137–1145, 1995.

[58] J. Kopecký, K. Gomadam, and T. Vitvar. hrests: An HTML microformat for describing restful web services. In *2008 IEEE / WIC / ACM International Conference on Web Intelligence, WI 2008, 9-12 December 2008, Sydney, NSW, Australia, Main Conference Proceedings*, pages 619–625. IEEE Computer Society, 2008.

[59] M. Koutraki, N. Preda, and D. Vodislav. Sofya: Semantic on-the-fly relation alignment. In E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, and K. Stefanidis, editors, *EDBT*, pages 690–691. OpenProceedings.org, 2016.

[60] M. Koutraki, D. Vodislav, and N. Preda. Deriving intensional descriptions for web services. In J. Bailey, A. Moffat, C. C. Aggarwal, M. de Rijke, R. Kumar, V. Murdock, T. K. Sellis, and J. X. Yu, editors, *CIKM*, pages 971–980. ACM, 2015.

[61] M. Koutraki, D. Vodislav, and N. Preda. DORIS: Discovering Ontological Relations in Services. In *BDA'15 - Journées de Bases de Données Avancées*, Ile de Porquerolles, France, 2015.

[62] M. Koutraki, D. Vodislav, and N. Preda. Doris: Discovering ontological relations in services. In S. Villata, J. Z. Pan, and M. Dragoni, editors, *International Semantic Web Conference (Posters & Demos)*, volume 1486, 2015.

[63] M. Koutraki, D. Vodislav, and N. Preda. Mapping Web Services to Knowledge Bases. In *BDA'15 - Journées de Bases de Données Avancées*, Ile de Porquerolles, France, 2015.

[64] R. Krummenacher, B. Norton, and A. Marte. Towards linked open services and processes. In A. Berre, A. Gómez-Pérez, K. Tutschku, and D. Fensel, editors, *Future Internet - FIS 2010 - Third Future Internet Symposium, Berlin, Germany, September 20-22, 2010. Proceedings*, volume 6369 of *Lecture Notes in Computer Science*, pages 68–77. Springer, 2010.

[65] J. Lafferty. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289. Morgan Kaufmann, 2001.

[66] V. Levenshtein. Binary Codes Capable of Correcting Deletions and Insertions and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[67] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the World-Wide Web conference*, 2003.

[68] W. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 1–12. Morgan Kaufmann, 1994.

[69] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33(1):49–84, 2000.

[70] W. Li, C. Clifton, and S. Liu. Database integration using neural networks: Implementation and experiences. *Knowl. Inf. Syst.*, 2(1):73–96, 2000.

[71] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3(1):1338–1347, 2010.

[72] S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, editors. *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 2014.

[73] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In K. Aberer, M. J. Franklin, and S. Nishio, editors, *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 57–68. IEEE Computer Society, 2005.

[74] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In Apers et al. [12], pages 49–58.

[75] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 77–88. Morgan Kaufmann, 2000.

[76] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 122–133. Morgan Kaufmann, 1998.

[77] P. Mitra, G. Wiederhold, and M. L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, pages 86–100, 2000.

[78] S. Muggleton. Learning from positive data. In *Selected Papers from the 6th International Workshop on Inductive Logic Programming*, pages 358–376. Springer-Verlag, 1997.

[79] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30:3–26, 2007.

[80] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *Proceedings of the 16th WWW*, pages 81–90, 2007.

[81] M. Oita, A. Amarilli, and P. Senellart. Cross-fertilizing deep web analysis and ontology enrichment. In M. Brambilla, S. Ceri, T. Furche, and G. Gottlob, editors, *VLDS*, volume 884 of *CEUR Workshop Proceedings*, pages 5–8. CEUR-WS.org, 2012.

[82] L. Palopoli, D. Saccà, G. Terracina, and D. Ursino. A unified graph-based framework for deriving nominal interscheme properties, type conflicts and object cluster similarities. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, September 2-4, 1999*, pages 34–45. IEEE Computer Society, 1999.

[83] L. Palopoli, D. Saccà, and D. Ursino. An automatic techniques for detecting type conflicts in database schemes. In *Proceedings of the 1998 ACM CIKM International Conference on Information and Knowledge Management, Bethesda, Maryland, USA, November 3-7, 1998*, pages 306–313, 1998.

[84] L. Palopoli, D. Saccà, and D. Ursino. Semi-automatic techniques for deriving interscheme properties from database schemes. *Data Knowl. Eng.*, 30(3):239–273, 1999.

[85] L. Panziera and F. D. Paoli. A framework for self-descriptive restful services. In L. Carr, A. H. F. Laender, B. F. Lóscio, I. King, M. Fontoura, D. Vrandecic, L. Aroyo, J. P. M. de Oliveira, F. Lima, and E. Wilde, editors, *WWW (Companion Volume)*, pages 1407–1414. International World Wide Web Conferences Steering Committee / ACM, 2013.

[86] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf.*, 2002.

[87] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[88] E. Peukert, J. Eberius, and E. Rahm. Rule-based construction of matching processes. In C. Macdonald, I. Ounis, and I. Ruthven, editors, *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 2421–2424. ACM, 2011.

[89] D. L. Phuoc, A. Polleres, M. Hauswirth, G. Tummarello, and C. Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 581–590. ACM, 2009.

[90] N. Preda, G. Kasneci, F. M. Suchanek, T. Neumann, W. Yuan, and G. Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In A. K. Elmagarmid and D. Agrawal, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 399–410. ACM, 2010.

[91] N. Preda, F. M. Suchanek, W. Yuan, and G. Weikum. SUSIE: search using services and information extraction. In C. S. Jensen, C. M. Jermaine, and X. Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 218–229. IEEE Computer Society, 2013.

[92] L. Qian, M. J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 73–84. ACM, 2012.

[93] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In Apers et al. [12], pages 129–138.

[94] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[95] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, 1995.

[96] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 251–261. ACM, 2016.

[97] M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014*. 2014.

[98] A. Segev and Q. Z. Sheng. Bootstrapping ontologies for web services. *IEEE Trans. Services Computing*, 5(1):33–44, 2012.

[99] P. Shvaiko and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Trans. Knowl. Data Eng.*, 25(1):158–176, 2013.

[100] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. K. Sellis. Top-k dominant web services under multi-criteria matching. In M. L. Kersten, B. Novikov, J. Teubner, V. Polutin, and S. Manegold, editors, *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 898–909. ACM, 2009.

[101] S. Speiser and A. Harth. Integrating linked data and services with linked data services. In G. Antoniou, M. Grobelnik, E. P. B. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Z. Pan, editors, *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*, pages 170–184. Springer, 2011.

[102] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB*, 5(3):157–168, 2011.

[103] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, 2007.

[104] M. Taheriyan, C. A. Knoblock, P. A. Szekely, and J. L. Ambite. Rapidly integrating services into the linked data cloud. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2012.

[105] C. Tatsiopoulos and B. Boutsinas. Ontology mapping based on association rule mining. In J. Cordeiro and J. Filipe, editors, *ICEIS (3)*, pages 33–40, 2009.

[106] P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2004.

[107] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *PVLDB*, 4(9):528–538, 2011.

[108] R. Verborgh, A. Harth, M. Maleshkova, S. Stadtmüller, T. Steiner, M. Taheriyan, and R. Van de Walle. Survey of semantic description of rest apis. In *rest: Advanced Research Topics and Practical Applications*, pages 69–89. Springer, 2014.

[109] W3C. Web services architecture. `https://www.w3.org/TR/2002/WD-ws-arch-20021114/`, 2002.

[110] W3C. Owl-s: Semantic markup for web services. `https://www.w3.org/Submission/OWL-S/`, 2004.

[111] W3C. Web services architecture. `https://www.w3.org/TR/ws-arch/`, 2004.

[112] W3C. Web service modeling ontology (wsmo). `https://www.w3.org/Submission/WSMO/`, 2005.

[113] W3C. Web service semantics - wsdl-s. `https://www.w3.org/Submission/WSDL-S/`, 2005.

[114] W3C. Xml path language (xpath) 2.0. `https://www.w3.org/TR/xpath20/`, 2007.

[115] W3C. Xsl transformations (xslt) version 2.0. `https://www.w3.org/TR/xslt20/`, 2007.

[116] W3C. Extensible markup language (xml) 1.0. `https://www.w3.org/TR/REC-xml/`, 2008.

[117] W3C. SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[118] W3C. SPARQL 1.1 query language for RDF. `http://www.w3.org/TR/sparql11-query/`, 2013.

[119] W3C. A json-based serialization for linked data. `https://www.w3.org/TR/json-ld/`, 2014.

[120] W3C. RDF 1.1 concepts and abstract syntax. `http://www.w3.org/TR/rdf11-concepts/`, 2014.

[121] W3C. RDF schema 1.1. `http://www.w3.org/TR/rdf-schema/`, 2014.

[122] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In Hencsey et al. [53], pages 187–196.

[123] J. Wang, J. Wen, F. H. Lochovsky, and W. Ma. Instance-based schema matching for web databases by domain-specific query probing. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 408–419. Morgan Kaufmann, 2004.

[124] S. Wang, G. Englebienne, and S. Schlobach. Learning concept mappings from instance similarity. In A. P. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. W. Finin, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings*, volume 5318 of *Lecture Notes in Computer Science*, pages 339–355. Springer, 2008.

[125] Linked open vocabularies. http://lov.okfn.org/dataset/lov/.

[126] H. Xiao and I. F. Cruz. Integrating and exchanging XML data using ontologies. 4090:67–89, 2006.

[127] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 412–420, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[128] Z. Zhang, B. He, and K. C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In G. Weikum, A. C. König, and S. Deßloch, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 107–118. ACM, 2004.