Domain-aware Ontology Matching on the Semantic Web

Master's Thesis

Kristian Slabbekoorn

Domain-aware Ontology Matching on the Semantic Web

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Kristian Slabbekoorn
born in Goes, the Netherlands

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`http://wis.ewi.tudelft.nl/`

Domain-aware Ontology Matching on the Semantic Web

Author: Kristian Slabbekoorn
Student id: 1228196
Email: kt.slabbekoorn@gmail.com

Abstract

The rise of the Semantic Web and Linked Open Data has led to a large number of different and heterogeneous datasets and ontologies published on the Web. This kind of heterogeneity of datasets has created a need to interlink them, i.e. to make explicit that two resources of different datasets in fact represent the exact same concept. This is the task commonly known as "ontology matching." This thesis presents work on the development and evaluation of a domain-aware ontology matching approach geared at interlinking domain-specific ontologies to the cross-domain ontology DBpedia.

We present a bootstrap method to automatically derive domain knowledge from an initial set of high confidence matches, and show several methods to translate this domain knowledge into an optimized filter that can be used to limit matches to. The domain derivation and optimization is accomplished in a fully unsupervised way. To explore the generalizability of the approach, we perform evaluations on two use cases: Last.fm artists and UMLS medical terms.

We show that for ambiguous data such as artist and band names, the proposed approach outperforms a traditional matching approach by as much as 17.1% on an F_1 scale of quality. Improvements for highly specialized data such as medical terms are less significant but can be gained by taking special measures. Finally, given an example application of medical training simulators, we show that the domain-aware approach is also potentially very useful when adapted to the semantic enrichment of free text, where a gain of as much as 26.6% is attained over traditional annotation approaches.

Thesis Committee:

Chair:	Prof. dr. ir. G. J. P. M. Houben, Faculty EEMCS, TUDelft
University supervisor:	Dr. L. Hollink, Faculty EEMCS, TUDelft
Committee Member:	Dr. M. Pinzger, Faculty EEMCS, TUDelft

Preface

Before you lies my Master's Thesis – the final work I've performed at TU Delft. When I started out this work around March 2011, I couldn't have imagined that this would be the point that I'd end. It took a while to find a suitable point of focus for the thesis, which basically started out with "So here's this medical simulator. Have fun with it!" A few months into the project, the work went into a very different direction than I'd initially imagined, but in the end I am quite satisfied with the result, and even managed to integrate things with what I'd originally started out with.

I would like to take this opportunity to thank my supervisor, Laura Hollink, as this result wouldn't have been possible without her support and guidance. She was always ready to provide comments and have meetings, pushing me into the right direction for my research. Secondly, I would like to thank my professor, Geert-Jan Houben, for allowing me to perform my thesis in the Web Information Systems group. Also thanks to Ilknur Celik, for helping me to get started in the initial stages of the project when I didn't know left from right, and my fellow Omega group members Peter, Martijn and Golnoosh, for the many comments and for bearing with my sometimes abhorrently lengthy presentation sessions. Lastly, a big thanks to Dr. Martin Pinzger for agreeing to be on my thesis committee.

I had the pleasure of attending the International Semantic Web Conference in Bonn, Germany in October of 2011, which has been an incredibly fruitful experience for me. I would like to thank Laura again for encouraging and helping me to submit a paper to the DeRiVE workshop held in conjunction with this conference. I would also like to thank the organizers of the workshop and the anonymous paper reviewers, for giving me the opportunity to present my work there and providing me with helpful comments, guidance and advice. My trip there would also not have been possible without the gracious financial support of Universiteitsfonds Delft, the Semantic Web Science Association (SWSA) and the Web Information Systems group here in Delft.

Last but not least, I would like to thank my friends and family for their continued support.

Happy reading!

Kristian Slabbekoorn
Delft, the Netherlands
January 19, 2012

Contents

List of Figures

Chapter 1

Introduction

The rise of the Semantic Web and Linked Open Data has led to a large number of different and heterogeneous datasets and ontologies published on the Web [11]. The aim of the Semantic Web is to give data on the Web explicit meaning in order to create a Web-scale, cross-domain, distributed knowledge base that is based on standardized protocols and languages. This greatly facilitates knowledge integration, as it no longer requires application designers to have to deal with specialized APIs individually developed by the various data providers on the Web – all data can be queried in a standard way (i.e. using the SPARQL language [43] to query RDF data [32]) and interpreted in a uniform way (i.e. reasoning over data structured with the OWL ontology language [33]). However, due to the open nature of the Web, in practice there tends to be little agreement between the semantics of datasets published in the Linked Open Data cloud. Reasons for this is that (1) they are often generated from legacy data that was previously stored in a relational database (possibly exposing an API for public access), thereby reusing the already established schemata for convenience when migrating to RDF, and (2) each publisher of data has their own view on how their semantics should look like. For example, one data publisher might classify the entity "Cat" as an "Animal," while another might prefer to more specifically classify a cat as a "Feline" instead. Similarly, given two botany-related datasets, one might have the entry "Narcissus" and the other "Daffodil." While a human might recognize that both are synonyms for the same flower, a computer will have the greatest difficulty determining this.

This kind of heterogeneity of datasets published on the Semantic Web has created a need to interlink them, i.e. to make explicit that two resources of different datasets in fact represent the exact same concept. Once such a link exists, computers may tell that these resources are the same, allowing them to leverage and combine properties of both resources and additional data from each respective dataset in order to produce new data. However, the discovery of such correspondences between entities of different ontologies or datasets is not trivial – it is in fact one of the major challenges of the Semantic Web [6]. The task is usually called "ontology alignment," or "ontology matching." Two or more ontologies are "aligned" with each other so that semantically similar entities may be merged.

A large number of ontology matching tools and methods have emerged in recent years [15]. A great majority of these systems focus the use of string similarity mechanisms in order to determine correspondences between entities of two different ontologies or datasets – the labels of both entities might be compared in a strict or fuzzy way, or additional properties such as aliases, translations, etc. could be considered. When matching entities with geographic properties, one could determine from the proximity between geographic coordinates whether e.g. two "City" entities refer to the same geographic location. Other than property comparison, one might turn to look at the schema in which entities in both ontologies are ordered, such as semantic classes and super-classes, or relations with other entities within the same ontology. For example, given the "Cat" example described earlier, the ontology that describes a cat as a "Feline" might have "Animal" as a super-class of "Feline," and through this super-class the correspondence with the "Cat" entity in the other ontology may be discovered. In other words, this type of approach to ontology matching exploits the hierarchy of the ontology in order to determine links between entities. However, it is often tricky to derive these types of correspondences, as there is always a string similarity matching step involved – in this case between the names of classes.

Another way is to restrict matches to the *domain* of entities we want to match, assuming these entities are grouped into classes that effectively describe and delimit a domain. Preferably, the domain of one dataset is derived in terms of schema information of the other, so that error-prone string similarity matching steps can be avoided. There has been little research into how one can include the domain of the data into the matching process in such a way. The main contribution of this thesis will be an exploration into the use of knowledge of the domain, and ways to derive it, to produce better or more matches.

We present a bootstrap method to automatically derive the needed domain knowledge from an initial set of high confidence matches, then show several methods to translate and optimize this domain knowledge into a filter that can be used to limit matches to. To explore the generalizability of the approach, we perform evaluations on several ontologies of different domains and with different properties. An early version of our approach and experiments has been published and presented in the Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web, held in conjunction with the 10th International Semantic Web Conference 2011 [48].

In order to somewhat limit the scope of our desired ontology matching approach, we will focus on the task of matching entities from domain-specific ontologies to a general-purpose, cross-domain ontology. The general-purpose ontology will be fixed to DBpedia [4]. We motivate these choices with the following arguments:

- DBpedia is the de facto centerpiece of the Linked Open Data cloud. It is large, widely used and constantly updated as it is based on Wikipedia – every addition or change made to Wikipedia will be reflected in DBpedia, either directly in the case of DBpedia Live [18] or periodically in the case of the standard DBpedia. This makes it an attractive choice as it will contain many concepts (over 3.64 million as of September 2011[1]) across various domains, ensuring a high probability that concepts from domain-specific ontologies will be included, thus giving ample opportunity for the creation of interlinks.

[1]urlhttp://blog.dbpedia.org/2011/09/11/dbpedia-37-released-including-15-localized-editions/

- Domain-specific ontologies, as the name suggests, are focused within a particular domain, and will generally only contain data restricted to that domain. However, it can be useful to integrate knowledge from other domains as well. For example, given a movie database such as IMDB[2] containing "Actor" entities, information on these actors will generally be focused on the movies in which they starred. If we want to know more about, for example, their place of birth, IMDB may contain the name of the city they were born, but not much else. Given a link to the DBpedia entry for the actor, we can find the city resource with specific information about this city (e.g. size, population numbers, etc.) linked to their place of birth. Furthermore, DBpedia is already interlinked with various popular datasets such as Freebase[3], Geonames[4] and so on, so by linking an entity to DBpedia, we indirectly also link it to these other datasets, further increasing the value of such interlinks.

- The act of ontology matching can be mapped to an optimization problem, where one aims to find the optimal configuration for a matching tool that provides the best possible matching, optimizing on some quality metric for the resulting alignment. The No Free Lunch theorem for optimization [52] states that no uniformly best solution exists for every problem domain – in fact, given the entire set of all problem domains, every optimization algorithm can be considered equivalent. In other words, some generality needs to be sacrificed in order to obtain a method capable of producing high quality matches. By focusing on a single target ontology, we aim to achieve a matching approach that yields high accuracy for the generated links, albeit at the cost of flexibility, as the approach will be somewhat limited in its application domain.

1.1 Research questions

The research questions that we will attempt to answer in this work are as follows.

1. ***How can we exploit the domain of a domain-specific ontology to improve alignment quality to a general-purpose ontology?***
   This will be the main theme of this thesis. We want to explore how and if domain information can be used to be able to perform higher quality matching. As mentioned, the target general-purpose ontology will be fixed to DBpedia. We need to consider what features to use as domain information, and how to effectively transform these features into a filter that can be used to determine whether a given target entity is included in or excluded from this domain.

2. *To what extent can such a domain derivation process be automated?*
   We would prefer a process that can derive the proper domain filter without any supervision at all. As it is not clear if – or to what extent – this is feasible, we need to explore the possibilities and impossibilities in terms of automation.

[2]`http://www.imdb.com/`
[3]`http://www.freebase.com/`
[4]`http://www.geonames.org/`

3. *What is the impact on alignment quality when ontology domains are taken into consideration?*
   If we can manage to produce a domain filter, we expect this filter to have a positive impact on the quality of matching compared to domain-unrestricted approaches. It needs to be tested whether this is true, and what the extent of this impact is, expressed in concrete quality measurements so that it can be effectively compared to approaches where a domain is not considered.

   a) *How is this impact affected when considering ontologies of different and/or broader domains?*
      Since we aim for an approach that is robust, it should ideally provide output of similar quality regardless of the domain-specific ontology we feed it. That said, matching quality cannot be expected to remain constant given domain ontologies of different general broadness – given a broader domain, the derived domain filter will be broader as well, effectively limiting its effectiveness. We need to test exactly how the devised approach will react to different domains, both in topic and broadness.

4. *How can real-world use cases benefit from this approach?*
   We want to give a concrete example of how a real application could use this approach to its benefit. The European ImREAL project[5], within which this thesis is executed, has provided us with a use case that consist of training simulators for the medical domain. Specifically, the focus is on psychiatric role-playing simulators where a user takes on the role of the psychiatrist and must apply his expertise in handling a patient with a mental disorder. We need to explore whether we can use the devised method to enrich the data contained in these simulators with links to DBpedia.

1.2 Research methodology

As mentioned previously, in this thesis we want to explore how domain knowledge can be used to produce better or more matches. We do this by developing an ontology matching system that leverages this type of domain information. We perform experiments to prove that the fundamental principles on which the method is built hold true, on two use cases: a dataset that consists of performing artists mined from music website Last.fm[6], and the UMLS, an expansive thesaurus-like dataset used in the medical domain [42]. Partial reference alignments are manually composed for both, so that quality of matching approaches can be evaluated by comparison of generated links with these reference alignments. Quality is determined in terms of precision, recall and F_1-score.

Two baseline, domain-unaware matching approaches and five different domain-aware matching approaches are devised. Differences between the latter five approaches are in the way the domain filter is derived and/or optimized. Output of each approach is assessed with the reference alignment, and ranked based on F_1-score in order to determine which approaches perform better than others. The experiments are done for

[5]`http://imreal-project.eu`
[6]`http://last.fm`

both use cases, so that performance of the different techniques under different circumstances can be evaluated as well. Additionally, we introduce an example application (ImREAL simulators) to evaluate how the approach can benefit the enrichment of data in a real-life case.

We will provide a physical implementation of the matching system, and use it to evaluate matching output for each technique. We conclude the research with a discussion of the results and an analysis of points of improvement that could be implemented in the future.

1.3 Organization of this thesis

The remainder of this thesis is organized as follows. We start with some background, expanding on the concepts and technologies that are used throughout the work, in chapter 2. This chapter should provide the reader with the fundamental knowledge concerning the field of the Semantic Web – specializing towards ontology matching – required to understand the motivations and considerations explained in further chapters. We additionally cover the use cases used to develop and evaluate the approach, and discuss some previously conducted work related to the area of domain-aware ontology matching.

The approach is explained in detail in chapter 3, and constitutes the main contribution of this work. The chapter starts out by explaining some preliminary matters related to tools used, and describes in brief the running example by which we try to make the explanation of our approach more understandable. The section that follows presents the domain-aware approach in a step-wise manner – the process is first presented as a pipeline of phases, and is followed by in-depth explanations of each phase in each of the subsections of the section. We end the section by presenting several optimization methods, to be compared in the evaluation chapter. The section that follows briefly expands on the actual technical implementation of the described approach, and the chapter is ended with a conclusion.

Chapter 4 contains the evaluation of the approach. We start by describing the setup and data used to perform the experiments with. Next, the actual strategy for evaluation is described. This includes the optimization methods devised during the approach and the concrete values we use for modifiable parameters of each of the methods. We present results and some possible refinements to the approach. The final section concludes the chapter and discusses the obtained results.

The thesis is ended with a general conclusion, summarizing the contributions made and discussing the answers to the research questions that we started the thesis with in section 1.1. Lastly, we present a list of future work that could be performed in order to improve upon the results, or gain more insight into the potential effects of domains on ontology matching.

A glossary of terms used throughout this thesis can be found in appendix A.

Chapter 2

Background

In this chapter, some general background information is provided on the concepts and theories that this work is fundamentally built on. The first section will give a brief introduction into the Semantic Web and semantic technologies, particularly Linked Open Data. Section 2.2 will delve further into the properties of data and their interrelations in the Linked Data cloud. This is followed by an explanation of the area of focus of this thesis, ontology matching, and its application in the Semantic Web field. Lastly, we will expand upon some work related to the advancement of the Semantic Web field, focusing on research conducted in the area of ontology matching.

2.1 The Semantic Web and Linked Open Data

The Web has been evolving continuously since its inception at CERN in 1991 by Tim Berners-Lee [9]. What started out as a simple means to share academic data by electronically linking together documents has grown to become a major part of our lives. With the advent of Web 2.0, the web gradually evolved into a fully interactive environment where users could not only access information from Web pages, but also publish their own content such as blogs and videos, stay in touch with their friends, chat/talk with people in real-time, do online banking, and so on. What remained missing from this picture, however, was the development of standard ways of representing data, and to give it universally understood semantics to allow even computers to understand what these pieces of data actually mean. Berners-Lee, realizing a need for this to happen (it had in fact been a part of his original vision of the Web) proposed his vision of a fully Semantic Web [10]. The Semantic Web is not intended to be a replacement for the current Web, but rather a new layer superimposed on top of it, injecting a new stack of technologies into the existing Web stack that are based on established Web standards and protocols such as HTTP and XML (see figure 2.1).

Since its initial proposition, various initiatives have been set up in attempt to realize the goal of creating a fully Semantic Web. One of the most successful initiatives has been “Linking Open Data” (LOD), which aims to interconnect vast amounts of data from many disciplines using standardized syntactic and semantic conventions, and have this data open for everyone to use and add to. In fact, the term “Semantic Web” is now often used interchangeably with “Linked Open Data.” The main goal behind LOD, or simply “Linked Data,” is to create a “Web of Data” next to the current

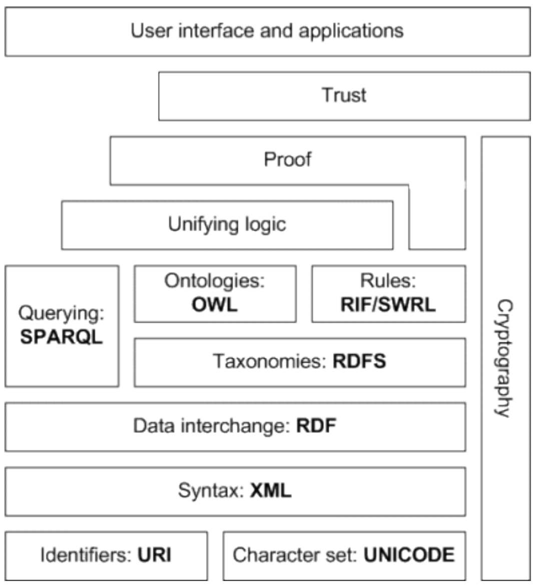


Figure 2.1: The Semantic Web stack as originally devised by Tim-Berners Lee.

"Web of Documents" – one that is interpretable by machines. In Linked Data, related data that was not previously linked is connected to form a new, large-scale, integrated knowledge base. It is this interconnection between datasets that is the most important aspect of Linked Data; by providing links to other datasets, applications may exploit this extra and possibly more precise knowledge. With standardized semantics, this knowledge base can be queried as one would a regular database.

Fundamental technologies used for the production of Linked Data include *HTTP*, the standard transfer protocol for traffic over the Web, *URIs*, used to identify data entities (as opposed to Web pages), and *RDF* [32], a graph-based data interchange format for representing the structured data (see figure 2.1). Originally RDF was an XML format, but several other formats have emerged. RDF will be further explained in section 2.1.1. Other components of the Semantic Web stack that bear explaining are *ontologies* and *taxonomies*, more popularly known as *vocabularies*, layered on top of RDF. Where RDF describes syntax in which to represent data, ontologies and vocabularies are meant to give semantics, i.e. concrete meaning, to this data. They are explained in more detail in section 2.1.2. Lastly, there is *querying* of Linked Data with the standardized query language *SPARQL* (section 2.1.3). The remaining components of the stack are outside the scope of this thesis and will therefore not be covered.

2.1.1 RDF

RDF [32], or Resource Description Framework, is a data model used for data interchange on the Web. It provides a standard way to describe things, and their relationships to other things. A "thing" can be any type of entity – a cat named "Felix," for example. In RDF, this is called a *resource*. Aside from things, other types of resources are kinds of things (i.e. classes), properties of these things, and values of those properties. Each RDF resource is represented by a unique identifier: a Web URI. A simple example that showcases all types of resources is shown in figure 2.2.

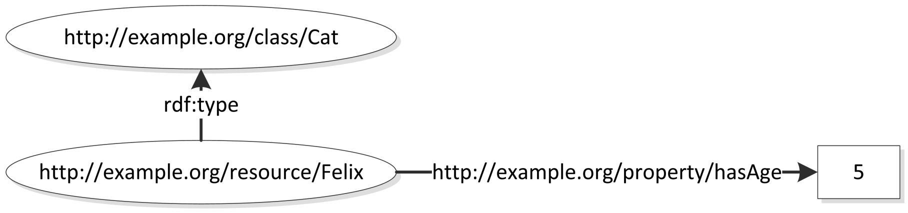

http://example.org/class/Cat
rdf:type
http://example.org/resource/Felix
http://example.org/property/hasAge
5

Figure 2.2: An illustrated RDF example.

In essence, RDF data is statements about resources as a graph of nodes and arcs, represented as a collection of *triples* in `<subject> <predicate> <object>` form. In the example of figure 2.2, we have the triple `http://example.org/resource/Felix rdf:type http://example.org/class/Cat`, which says that Felix is an entity of type Cat. `rdf:type` is a property resource, where `rdf` is a *prefix*, or shorthand notation, for `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, where the RDF vocabulary is defined. `type` is a core RDF property reserved to point to the class of an entity.

The RDF vocabulary – and its extension, RDFS, which allows describing taxonomies of classes and properties – are limited to very fundamental properties, but other ontologies and vocabularies can be built on top of it (see section 2.1.2). The second triple from the example, `http://example.org/resource/Felix http://example.org/property/hasAge "5"`, says that Felix is aged 5 through the property `hasAge`. This property is an example of one that is not pre-defined in RDF, but part of the proprietary ontology that describes the data of `http://example.org`.

While "RDF" in general refers to the data model, there are several possible *serializations* with which to represent this model, such as RDFa (used for embedding RDF into HTML), Turtle [5] and N3 [7]. These serializations provide ways in which to represent triples. When a URI pointing to an RDF resource is *dereferenced* (i.e. accessed) by an application, the triples representing this resource and its properties should be returned in one of the serialization formats, so that the application can process this data.

2.1.2 Ontologies and vocabularies

An *ontology*, in the philosophical sense of the word, concerns "what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences[1]." In the context of information science, an ontology is the representation of this concept in the form of a structural framework for organizing information. Ontologies are employed in a multitude of disciplines, such as artificial intelligence (AI), knowledge representation, enterprise architecture and the Semantic Web. A *vocabulary* is very similar to an ontology – in fact, in the context of the Semantic Web, there is no real clear distinction between the two. That said, the word "ontology" tends to be associated with more complex and formal collections of terms, whereas the word "vocabulary" is usually used when strict formalism is not employed.

[1]`http://en.wikipedia.org/wiki/Ontology`

An ontology can be divided into two parts: the schema and the entities. The schema is the collection of classes and properties that say something about the entities and their interrelations. On the Web, such an ontology is often called *dataset*. In this thesis, we will use “ontology” and “dataset” interchangeably.

OWL Ontologies are composed with ontology languages. The standard language describing the fundamental semantics for datasets on the Semantic Web is the Web Ontology Language (OWL) [33]. It is built on top of RDF, extending it to allow formal descriptions of the meaning of terminology used in Web documents, and definitions of additional constraints on the structure of RDF graphs. OWL defines the basic framework for structuring data in a hierarchical way. It defines three types of things: classes, properties and individuals. These overlap with the different types of RDF resources. A class is a group of individuals that belong together because they share some properties (e.g. `http://example.org/class/Cat" rdf:type owl:Class`). Properties are used to state relationships between classes or individuals (e.g. `Cat rdfs:subClassOf Feline`, `Narcissus owl:sameAs Daffodil`). Individuals are instances of classes (also called “entities” or simply “instances”).

Common vocabularies Outside of OWL, there are a great number of supplementary vocabularies that help describe entities and their relationships in a variety of domains. Commonly used vocabularies include Dublin Core [50], for describing metadata about resources, FOAF (friend-of-a-friend) [12], used to model users and user communities on the Web, and SKOS [21], which provides a model for expressing the basic structure and content of concept schemes such as thesauri and classification schemes. Additionally, effort is being made on unifying and standardizing vocabularies designed to model time, events, media, and so on.

While a data publisher is generally allowed to create his own vocabulary terms for properties and give them any meaning he wants, in the interest of interoperability it is considered good practice to use properties of existing, established vocabularies wherever possible.

2.1.3 SPARQL

The main vision of the Semantic Web and Linked Data is essentially to create a Web-scale, distributed database that can be queried as one would a regular database. This requires a query language that can handle the graph-based nature of RDF.

The most commonly used query language for relational databases is SQL. Loosely based on the SQL grammar is SPARQL, which allows for querying RDF [43]. With SPARQL, we can retrieve the data we need from the Linked Data cloud using SPARQL endpoints, which serve RDF graphs. An example query could look as follows:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?cat ?age FROM <http://example.org> WHERE {
    ?cat rdf:type <http://example.org/class/Cat> .
    ?cat <http://example.org/property/hasAge> ?age .
}
```

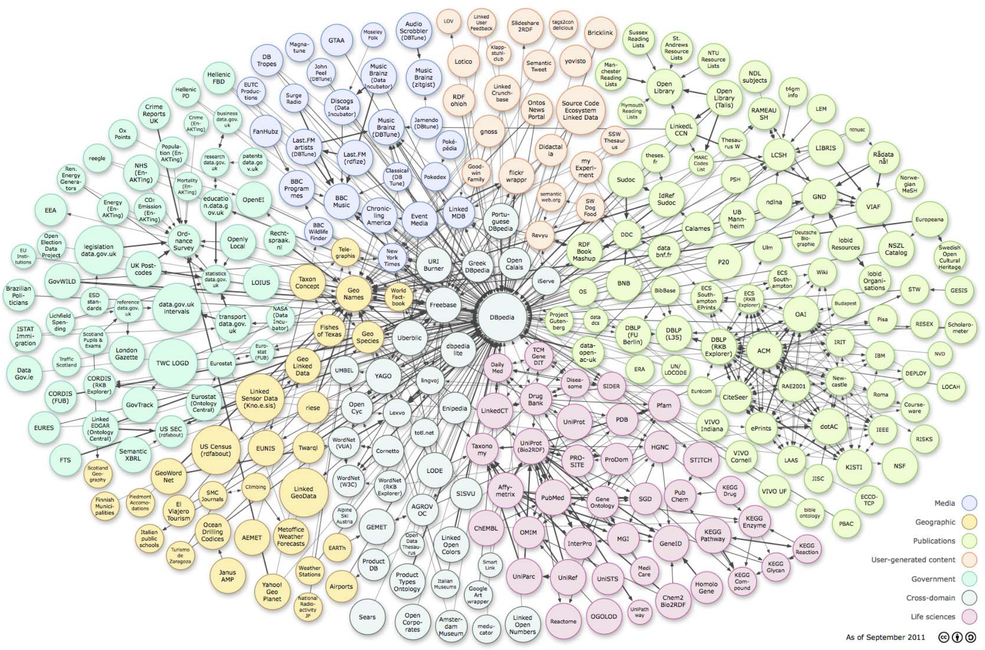


Figure 2.3: The state of the Linked Open Data cloud as of September 2011.

This query will return all entities of type “Cat” along with their age, as a set of records much like what one would obtain as output of an SQL query on a relational database, but it is also possible to return actual RDF graphs (using the `CONSTRUCT` keyword in the place of `SELECT`). When entities of different datasets are interlinked, for example with the `owl:sameAs` property, we can query for this property on one dataset and obtain the entities of the other. From there, we can query further, for properties that are contained in this second dataset. Herein lies the power of the Linked Data principle. However, before we can take advantage of these kind of correspondences between datasets, they need to be actually created first.

2.2 Linked datasets

A graph-representation of the present state of the Linked Data cloud is shown in figure 2.3[2]. It is periodically updated to include newly published datasets[3]. Each circle in the diagram is a dataset – the size of the circle is representative for its size. Shown in the center of the image is DBpedia[4], illustrating its role as centerpiece of the Linked Data cloud. Each color represents a different general domain, such as media, government data, medical data, etc. The arrows indicate that there are at least 50 interlinks from one dataset to another; the thicker the arrow, the more interlinks that exist between them.

[2]Retrieved from: `http://richard.cyganiak.de/2007/10/lod/`.

[3]`http://linkeddata.org`

[4]`http://dbpedia.org`

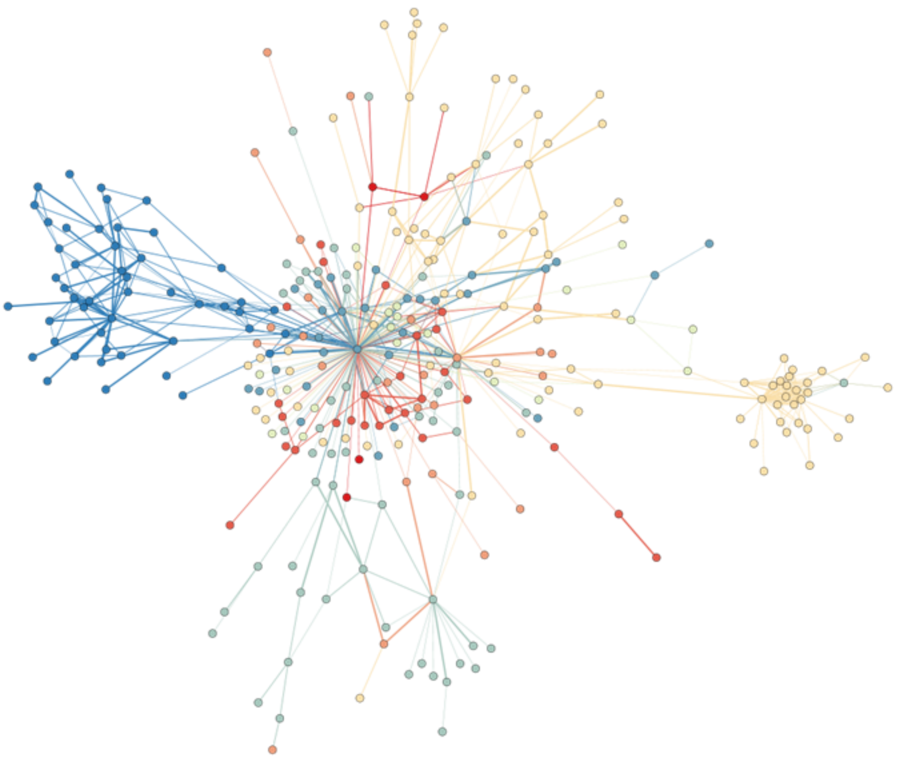


Figure 2.4: A network analytic view of the Linked Open Data cloud.

The graph is however somewhat deceptive in that it makes it look like the Linked Data cloud is homogeneous and evenly structured. In reality, it is actually quite clustered. Figure 2.4 shows a network analytic view of the same Linked Data cloud [16]. It can be seen here that links are not evenly spread out over the datasets, but focused inside a select few clusters that are mutually interlinked.

At the time of writing, the full Linked Data cloud is estimated to contain around 30 billion triples, with only 500 million of them being links. Of these links, around 330 million are trivial, such as links between unique IDs in medical datasets, and several dozen million are near trivial, such as links between DBpedia and Freebase based on correspondences between Wikipedia links associated to entities [51]. This leaves a very generous estimate of at most 100 million interlinks between datasets of real value, which is around 0.3% of the total amount of triples. This is a very small amount, and shows a need for more interlinking of datasets.

The following sections describe what is needed to know about the structure of two datasets that we make use of in this thesis: DBpedia and Freebase.

2.2.1 DBpedia

For this work we have chosen to focus on the DBpedia dataset, a centerpiece of the Linked Data cloud [4]. DBpedia can be regarded as the Linked Data equivalent of Wikipedia – in fact, all information contained in this dataset has been automatically extracted from Wikipedia. It is therefore a type of mirror for Wikipedia – it does not produce data of its own, but simply structures data extracted from Wikipedia pages. This structuring is done in RDF, according to an ontological schema of classes and properties, which has been taken and adapted from infoboxes on Wikipedia. Measures have been taken to clean this data by merging instances where there are multiple infoboxes for the same class, or different property names for the same property. A static version is available for download, and a SPARQL endpoint has been provided for online access.

For each Wikipedia page, there is a resource in DBpedia. This resource does not contain the full text of the page, but only structured information. Aside from infobox classes and properties, it contains an abstract for the resource (the first paragraph of the corresponding page), which disambiguation pages and redirects point to this resource, and external links to other large datasets such as Geonames and Freebase.

DBpedia resources can be classified by a number of different class hierarchies, which we cover below.

DBpedia classes DBpedia classes are the official DBpedia ontology class types. They are extracted from the most common infobox types on Wikipedia. The result is a shallow, cross-domain class hierarchy for structuring DBpedia resources that consists of around 320 classes. Not all DBpedia resources are categorized according to a class – only if the resource in question had an infobox on Wikipedia, as data in DBpedia is fully dependent on Wikipedia.

Wikipedia categories DBpedia resources are also be classified according to categories. Categories are taken as-is from Wikipedia, where any category can be assigned to any page by the users. The categories themselves are organized in a hierarchy. This hierarchy is non-strict and uncontrolled, as super- and subsumption relations can be added at will.

YAGO YAGO is a large ontology comprising entities and relations that has been automatically derived from Wikipedia and WordNet [49][36]. Its main components are a taxonomic `is-a` hierarchy similar to the DBpedia classes and semantic relationships between these entities. There are nearly 3 million YAGO classes assigned to DBpedia resources. The facts for YAGO have been automatically extracted from the category system and the infoboxes of Wikipedia and have been combined with taxonomic relations from WordNet. It has been shown to have an accuracy rate of 95%.

2.2.2 Freebase

Freebase[5] is another knowledge base that is available as Linked Data (i.e. in RDF). It calls itself "*an entity graph of people, places and things, built by a community that loves open data.*" It is intended to be a general-purpose encyclopedia much like Wikipedia. It is different from DBpedia in that DBpedia does not produce any data of itself, but merely "mirrors" Wikipedia, while Freebase is its own knowledge base, integrating data from various sources, as well as allowing the community to edit anything they like.

[5] `http://www.freebase.com`

Freebase has its own typing system, although unlike the DBpedia classes/categories and YAGO classes, it only has two levels and is thus a practically flat hierarchy. For example, there is a class `/music` which has subclasses `/music/artist`, `/music/musical\_group`, `/music/group\_member`, and so on, but there is no level deeper than the second. Links exist between DBpedia and Freebase for over 3 million entities, through which these types are also indirectly assigned to DBpedia entities.

2.3 Ontology matching

Ontology matching, also known as *ontology alignment*, is one of the major challenges of the Semantic Web [6]. Links from one resource to another can greatly increase the value of the data in terms of better interpretation, connection of applications, and enrichment of the data.

The need for ontology matching has arisen from the need to integrate heterogeneous databases or datasets that are each defined with their own particular vocabulary. A distinction can be made between *schema matching* and *instance matching*. Schema matching focuses on finding correspondences between schema-level entities of ontologies or vocabularies, such as classes and properties, while instance matching refers to the process of finding correspondences between entities of two datasets. They are not mutually exclusive, as schema matching is often used to assist in the task of instance matching. In a Semantic Web context, this strict distinction is often omitted, and "ontology matching" can refer to either or a combination of both.

The terms ontology alignment and ontology matching are also often used interchangeably, although Euzenat and Shvaiko [15] define ontology matching as the process of finding correspondences between two or more heterogeneous ontologies, and an ontology alignment as the result thereof. We choose to adopt these definitions for this thesis.

Definition 1 (Ontology matching)**.** *The process of finding relationships or correspondences between entities of different ontologies.*

Definition 2 (Ontology alignment)**.** *A set of correspondences between two or more (in case of multiple matching) ontologies (by analogy with molecular sequence alignment). The alignment is the output of the matching process.*

A successful application of ontology matching is especially crucial for the Semantic Web, as it an indispensable factor for the ensured growth and increased integration of the Linked Data cloud, which needs to be possible in a scalable, i.e. largely automated way. However, ontology matching is a non-trivial task, as it can be very difficult to determine exactly when two resources are the same. Consider, for example, the named entity "Apple" that we want to match to the corresponding DBpedia resource. Depending on the context in which this word is used, it could refer to an apple (the fruit) or Apple Inc. (the company), or a band named Apple, or any of the remaining twenty or so senses that are listed in the disambiguation page for "Apple" on Wikipedia[6]. In other words, we need some way to determine the correct sense of

[6] `http://en.wikipedia.org/wiki/Apple_(disambiguation)`

a word. This task is called “word sense disambiguation” (WSD), one of the major challenges in artificial intelligence (AI) [20].

For instance matching, entity disambiguation is fundamentally performed by defining (1) some (complex) similarity function – usually based on string similarity measures – in order to determine for each source entity a measure of similarity to target entities, and (2) a similarity threshold by which to determine whether a link should be made or not. This similarity threshold is not necessarily an absolute similarity value, but can be a complex function as well, i.e. making use of the schema to check for containment or non-containment in some class.

2.3.1 Alignment evaluation

In order to determine the quality of an alignment (i.e. set of links) generated for a complete set of entities by an ontology matching system, it is usually evaluated by comparison to a manually created set of links for a small sample set of the entities. This manually created sample linkset acts as a ground-truth for the dataset, and is usually called a *partial reference alignment*, or simply *reference alignment*. If the sample size is large enough, then by the law of large numbers it can be assumed that the level of matching quality according to this partial reference alignment will lead to the same level of matching quality when applied to the entire set of entities. A reference alignment can be formally defined as follows.

Definition 3 (Reference alignment)**.** *A (partial)* reference alignment *R is a sample set of links from entities in a source ontology to all matching entities in a target ontology that have been confirmed – preferably by domain experts – to be correct. That is,*

$$R = \{\langle e_s, E_t \rangle_i : 1 \leq i \leq n, e_s \neq \emptyset\} \tag{2.1}$$

where e_s is an entity in the source ontology, E_t a set of matching entities in the target ontology, and n the sample size. E_t may be empty for some e_s.

To get a concrete measure of the quality of a matcher, the information retrieval concepts *precision*, *recall* and *F_1-score* are often applied. Originally these measures were designed to determine the quality of document search engines, but it is also applicable to ontology matching. In this context, they can be defined as follows:

Definition 4 (Precision)**.** *In the context of ontology matching,* precision *is the ratio of correctly generated links to all generated links, according to a reference alignment. i.e.*

$$Precision = \frac{|\{reference\ matches\} \cap \{generated\ matches\}|}{|\{generated\ matches\}|} \tag{2.2}$$

Definition 5 (Recall)**.** *In the context of ontology matching,* recall *is the ratio of correctly generated links to all links included in a reference alignment. i.e.*

$$Recall = \frac{|\{reference\ matches\} \cap \{generated\ matches\}|}{|\{reference\ matches\}|} \tag{2.3}$$

Definition 6 (F_1-score)**.** *The F_1-*score *is the harmonic mean between precision and recall.*

$$F_1 = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)} \tag{2.4}$$

The subscript $_1$ in F_1 stands for an even weighting in precision and recall, as opposed to e.g. $F_{0.5}$ for more emphasis on precision and F_2 for more emphasis on recall. The harmonic mean is different from the common, arithmetic mean in that the mean between precision and recall gets an increasing downwards bias as both values grow increasingly far apart.

2.3.2 Matcher tuning and optimization

The act of ontology matching can essentially be mapped to an optimization problem: we aim to find the optimal configuration for a matching tool that provides the best possible matching, optimizing on some quality metric (e.g. F_1-score) for the resulting alignment. The configuration of a matching tool consists of the selection of a matcher, i.e. the heuristics used in order to determine whether two entities should be linked or not, as well as the tuning of configurable parameters for these heuristics, such as thresholds, filter compositions, and so on. Configurability is all but required if we want a matching tool to work well for more than just a single situation, as no two datasets are the same, so parameters will need to be changed to obtain optimal performance. This is especially true in a dynamic and heterogeneous setting such as the Web. Ideally, the optimal configuration is found in a fully unsupervised way, but such automated matcher selection and self-configuration is a challenge [47].

For a given matcher, the search for the optimal parameter settings for them given some dataset is known as *matcher tuning*. This may involve run-time reconfiguration of parameters and thresholds, in order to find the most appropriate values for the dataset. The quality of a setting is usually tested on small set of manually created sample matches. It is a difficult optimization problem as search spaces, i.e. possible parameter/threshold combinations, can be extremely large.

2.4 Use cases

The experiments for this work are performed on two different use cases. They are datasets chosen according to a number of criteria that we demanded them to meet:

- The datasets should be large enough so that experiments performed on them are statistically meaningful;
- The datasets should be well-known and commonly used, so that the outcome of this work may have actual “real-world” value;
- The two datasets should be sufficiently different from each other: not only in terms of their topics, but also in terms of the ambiguity of their entities.

Based on these criteria, we ended up choosing to process one dataset from the domain of events, namely performing artists mined from the music website Last.fm made available by the EventMedia[7] project; and one dataset from the biomedical domain, the UMLS. The first criteria is met quite evidently; the Last.fm Artist dataset contains over 50,000 entities, the UMLS contains millions of terms. Both are also in

[7] `http://eventmedia.cwi.nl`

widespread use; Last.fm is a popular music site with millions of registered users and the UMLS is widely used in the field of medicine. Lastly, they are also very different in the aspects that we were looking for. Aside from the major difference in topics, artist names tend to be common and therefore very ambiguous. For example, there is a musical artist named "Scott Miller" in Wikipedia, but there is also an entrepreneur, an author, a footballer and a swimmer that are all named "Scott Miller." This ambiguity will allow us to effectively test and demonstrate to which extent our approach can facilitate concept disambiguation. On the other hand, the UMLS medical terms tend to be much more specialized in nature and thus less ambiguous, making it interesting to see if and to what extent our approach could yield an improvement for such a dataset.

The two sub-sections that follow will delve deeper into the characteristics of each dataset.

2.4.1 EventMedia Last.fm dataset

The EventMedia dataset is composed of events and media descriptions associated with these events [26]. It is represented in RDF format. It contains three classes of entities: agents (e.g. artists), venues and events. All data contained in this dataset has been mined from three large public event directories:

1. Last.fm, a popular service for people to track and share music they listen to. The system builds a detailed profile of the users' taste in music and offers recommendations based on this profile. Outside of artist/album listings, Last.fm also offers details about past and upcoming events.

2. Eventful[8], a free service that offers listings of entertainment and live events. It covers a wide range of event types, from movies, concerts and sports to nightlife.

3. Upcoming[9], an event register similar to Eventful, run by Yahoo!.

The data contains some links to other Linked Data sets, such as DBpedia, Freebase and Geonames, but they are very sparse. We choose to focus on Artist entities for reasons mentioned earlier. Since there are 50,151 Artist instances in the Last.fm dataset, 6543 instances in the Eventful dataset, and none in the Upcoming dataset, it makes sense to take the largest set to experiment with. In the EventMedia schema, each Artist entity is identified by a unique ID string and is represented as an instance of FOAF class `foaf:Agent`, where the name of the artist is contained in the property `rdfs:label` (required) and a description of the artist is contained in Dublin Core property `dc:description` (optional). See figure 2.5. A select few of the Artist entities also contain links to external datasets such as DBpedia in the form of `owl:sameAs` properties, but we ignore these for our purposes.

We have created a reference alignment containing manually determined links to DBpedia resources for 1000 Artist entities. The 1000 entities have been selected at random from the source dataset of 50,151 artists. Not all artist entities have links to DBpedia (only around 36%), and some entities have labels that contain multiple artists, and therefore have more than one link to DBpedia. Since Last.fm maintains

[8] `http://eventful.com`

[9] `http://upcoming.yahoo.com`

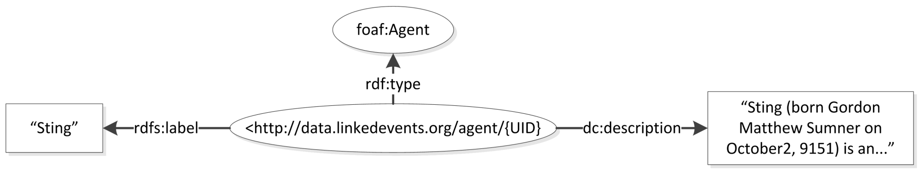

foaf:Agent
rdf:type
"Sting"
rdfs:label
<http://data.linkedevents.org/agent/{UID}>
dc:description
"Sting (born Gordon Matthew Sumner on October2, 9151) is an..."

Figure 2.5: A schematic representation of Artist (formally Agent) entities in the EventMedia dataset.

artist names as IDs for pages, it happens sometimes that multiple artists with the same name share a page. In this case, we do not expect links to each artist, but any one will be considered correct.

2.4.2 UMLS dataset

The UMLS, or Unified Medical Language System, is a combined medical thesaurus developed by the U.S. National Library of Medicine (NLM), designed to bring together various health and biomedical vocabularies and standards to enable interoperability between computer systems operated in the field of medicine [42][28]. The UMLS officially consists of three components:

1. The UMLS Metathesaurus, which contains codes, terms, definitions and descriptions from many different medical vocabularies
2. The UMLS Semantic Network [41], which is a hierarchy of classes (semantic types) and their interrelationships
3. A set of natural language processing tools called SPECIALIST

The first two components will be used in this work, where each class of the Semantic Network will be considered the domain of the terms that it contains. The UMLS' thesaurus-like structure makes it somewhat comparable to WordNet [36], a frequently-used, more general terminological system, although there are some clear differences as well [1]. In particular, outside of covering a much more specific domain, the UMLS records all strings provided by medical vocabularies for a given term; this includes for example inflectional variants and case and hyphen variants. Furthermore, inter-concept relations in UMLS are not always defined with precision. Hierarchical relationships that exist between terms directly, use whatever principle the underlying vocabulary it was based on uses, so they are not always consistent and reliable. We will therefore not use these. The UMLS Semantic Network was in fact designed to overcome this by classifying terms from the various vocabularies according to a class hierarchy independent of the vocabularies they originated from.

The UMLS is not originally a dataset designed for use on the Semantic Web. However, there exists an RDF conversion of the UMLS in the form of Linked Life Data [2] (LLD), a platform for semantic data integration for the biomedical domain, integrating a wide range of biomedical datasets, such as UniProt [53], PubMed [39] and Entrez-Gene [31], as well as taxonomies such as MeSH [29]. LLD is connected to the Linked Data cloud and offers a publically available SPARQL endpoint for querying the data.

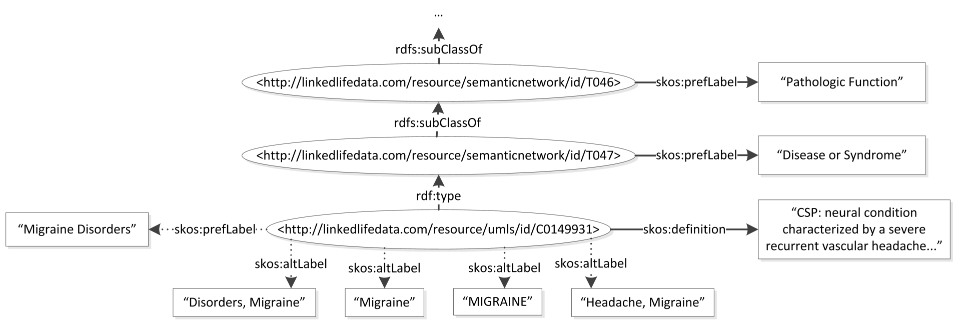


Figure 2.6: A schematic representation of UMLS terms in the Linked Life Data dataset.

The UMLS dataset in LLD is limited to vocabularies with the Category 1 and Category 2 NLM license type (e.g. SNOMED [40] and other high-quality resources are omitted because of NLM's strict licensing policy) and has been represented with the SKOS RDF vocabulary [21]. This is the source we use to obtain the needed UMLS data for the evaluation of our approach. Each UMLS term in LLD is identified by its UMLS UID, and is represented as an instance of the corresponding Semantic Network class. Names of terms are represented by properties `skos:prefLabel` and `skos:altLabel` – there is always one preferred label (i.e. the official term name), and there can be 0 or more alternative labels (e.g. inflectional variants and case and hyphen variants) for each term. Term definitions are represented by the property `skos:definition` and may not always exist. See figure 2.6 for a graphic example of how a term is structured. Note that in this figure, label properties for the UMLS resource are given dotted arrows. This is because they are implicit; each label is in fact a separate resource which includes not only the label string, but also extra providence information such as the source vocabulary the label originated from. This has been abstracted from the diagram for readability reasons and because we do not make use of this information.

Appendix B shows the full class hierarchy of the UMLS Semantic Network. Each class can effectively be considered a (sub-)domain. Since the full dataset is rather broad, we opt to limit ourselves to sub-domains. One thing we would like to test for our approach is exactly how deep into the hierarchy we need to go for it to gain a definite improvement over domain-less matching, and how subsequently selecting classes higher up the hierarchy as domain will affect quality of the matching. Two attractive classes that match our first criteria are "Physiologic Function" and "Pathologic Function," which are two disjoint classes on the same level of the hierarchy (i.e. sibling classes). We will test performance of our approach on these two classes separately first, then check how selecting a common superclass, "Phenomenon or Process," will affect final performance.

For UMLS, we have created two reference alignments of 500 entities each for the Physiologic Function and Pathologic Function classes. The Phenomenon or Process class can then be evaluated by taking the union of these reference alignments.

2.5 Related work

This section will discuss some of the work related to the main topics handled in this thesis. There has been a great amount of research done into the topic of ontology matching from the perspective of a variety of fields, such as linguistics, AI and knowledge management. Various tools and algorithms have emerged that automate the task of matching two ontologies or datasets. Most of these systems use string similarity measures and/or structural measures to determine the similarity between a pair of resources. In light of the sheer volume of related work, we split up the related work in a number of specific topics and focus on the research that involves either (1) the matching of entities to DBpedia, or involves the use of DBpedia as auxiliary knowledge base, (2) the exploitation and incorporation of domain knowledge into the linking process, or (3) the parameter optimization of matchers.

Matching that involves DBpedia In [34], a supervised machine learning approach for mapping search queries to DBpedia concepts is proposed. Queries are an example of a situation where no source context is available for calculation of similarity of concepts, and therefore need to be handled differently. The approach first gathers candidate concepts by searching for each query term in DBpedia. Second, a pre-trained classifier based on machine learning algorithms decides which candidates are relevant or irrelevant, and ranks them according to relevance. The algorithm achieves better results when features pertaining to a user's search history are taken into account. This work is similar to our situation in that we also try to match terms that may not always include context information, but we do not have information such as search histories available. Furthermore, such a supervised machine learning approach is very labor-intensive and requires various resources to set up, which is something we prefer to avoid.

The work described in [27] matches entities from a database of the BBC to DBpedia by finding candidates in DBpedia based on string-similarity and wiki inlink metrics, then comparing classifications of their own entities to the classes and categories of the DBpedia candidate resources to derive correspondences. In our case, we do not assume to have a classification of the source data available.

Several methods have been proposed that leverage additional knowledge from DBpedia to derive links between arbitrary Linked Data sets. BLOOMS+ [23][24] is an ontology matching system that uses the Wikipedia category hierarchy to bootstrap the process of finding schema-level links between Linked Data sets. We exploit the Wikipedia category hierarchy in a similar fashion; not to find matches directly but to find categories (and classes) that effectively describe our domain of interest.

DBpedia Spotlight [35] is an open source *named entity recognition* (NER) tool that detects mentions of DBpedia resources in natural language text (i.e. named entities), then adds links to these resources in the text. For each possible mention, or *surface form*, candidate resources are gathered by looking for string matches between the surface form and the following features in Wikipedia/DBpedia: page titles, redirect page titles, disambiguation page titles, anchor texts of links within Wiki pages to other pages and DBpedia name properties. The tool additionally allows users to define filters in terms of a selection of DBpedia classes and/or Freebase classes, or in terms of a SPARQL query; it does not leverage domain information to assist in the disambiguation process of itself. Several other well-known named entity recognition tools are OpenCalais[10], Zemanta[11] and AlchemyAPI[12]. These three are all commercial systems, but provide limited free access. They are not specifically created for use in a Linked Data context, but can often be mapped to DBpedia resources (if a Wikipedia URL is provided) or other Linked Data sources. All three systems are based around the concept of comparing context similarity between source text and target concepts, and provide entity types (sometimes based on a proprietary ontology, but mappings to DBpedia types are provided by [45]) and URIs. A survey and comparison tool for comparing these and more NER systems can be found in [45].

For the approach described in this thesis, we have taken DBpedia Spotlight as a starting point to assist with the creation of matches (see section 3.1). Although it is not shown to be the best performing NER tool out-of-the-box, its high degree of configurability and extensive focus on DBpedia make it an attractive foundation around which to build a more comprehensive approach.

Exploitation of domain knowledge The authors of [3] attempt to use both domain knowledge and multilingualism to match concepts in a lexicon to DBpedia. They perform the disambiguation of concepts in a number of steps. First, for a (stemmed) direct string match from source concept label to DBpedia resource label, they check whether there are concept labels in different languages in the source concept that also have a direct string match to the label in the corresponding language in DBpedia. If this is the case, a match is made if any of the following two situations hold: (1) if the DBpedia label includes a term in brackets (its "domain") that matches the string of one of the source concept classes; (2) one or more terms associated with concepts of the same class as the source concept occur in any of the DBpedia properties `dbpedia-owl:abstract`, `dcterms:subject` (i.e. the category of a resource), `rdfs:comment` or `dbpedia-owl:wikiPageRedirects`. Domain knowledge is thus exploited, but a heavy reliance on purely string similarity matching can lead to significant inaccuracies in the result. Our approach is different in that we derive and exploit domain knowledge without having to rely on any string similarity measures.

Kalyanpur et. al. [25] describe a type coercion framework for use in a question answering scenario; in this case for IBM's Watson [19]. Their system makes extensive use of DBpedia and domain knowledge extracted from DBpedia. *Typing* is the task of recognizing whether a given concept is a member of a given class, a fundamental problem in the field of AI. The authors solve this by a so-called *generate and type* approach, where they first generate a set of candidates based on a given question to be answered, then check whether these candidates can be "coerced" to the type of the question. The question type is first produced by lexical analysis. Next, candidate answers (strings) are generated by querying knowledge bases based on information gathered in the question analysis stage. They are then scored using various knowledge sources and ranked based on a variety of features using machine learning algorithms. DBpedia is one of the prime knowledge bases used for candidate scoring. For every candidate answer string with the question as context, candidate resources are gathered by looking

[10]`http://opencalais.com`
[11]`http://www.zemanta.com`
[12]`http://www.alchemyapi.com`

for string matches between the answer string and features in Wikipedia/DBpedia such page titles, redirect page titles and disambiguation page titles, i.e. very similar to how DBpedia Spotlight does this. Subsequently, the candidates are ranked based on properties of the listed features and context similarity measures. The next step is to determine whether the type of each candidate resource corresponds to the question type. This is done by analysis of the DBpedia classes, YAGO classes and Wikipedia categories and lists of each candidate resource. As each class/category directly assigned to a resource is usually highly-specific, candidate classes are generalized to improve coverage by moving up the class/category hierarchies a certain number of levels to obtain a class that is sufficiently general. The labels of these general classes are then aligned with the candidate answer string by a complex similarity algorithm, which takes into account not only lexical but also structural information.

For our approach, we obtain candidate DBpedia resources in much the same way, and similarly leverage the class/category hierarchies to improve the quality of matching. The main difference is that the Watson system attempts to find the correct candidate class by complex string similarity measures and hierarchy-based alignment with the source (question) type, while we do not assume to have any source type information available and determine the correct target candidate class by bootstrapping with a sample of source entities. While this requires there to be many source entities that belong to the same domain (a requirement that does not hold for a question answering system such as Watson), it allows us to avoid having to rely on error-prone similarity algorithms for domain derivation.

Tuning parameters of matchers The authors of [22] present an approach to automatically generate linkage rules from a set of reference links. Linkage rules are based on which comparison methods to use to determine correspondences between entities, such as different string similarity metrics or distance measures, and which parameters these methods should ideally have. The reference links have been manually created. A genetic algorithm is applied to learn which rules and parameters work best on a dataset, based on performance according to a partial reference alignment. They show that the automatically learned rules achieve similar accuracy to those manually created by humans.

In [44], ECOMatch is proposed. ECOMatch is a generic parameterization system designed for use on top of any ontology matching system. It asks the user to provide example mappings instead of parameter settings for the matcher, then automatically determines a suitable configuration based on those examples. They show that a correlation exists between scores of a configuration according to a portion of a reference alignment and a full reference alignment, motivating that a small set of example matches will be enough to optimize a configuration for the full ontology on. They show that their approach can yield improvements over the default settings for two well-known ontology matchers.

Some aspects of our approach are similar to these two works. Like the first work, we also use a genetic algorithm for optimization, but rather than learning linkage rules to find correspondences, we learn a domain filter to restrict correspondences to. ECOMatch is also different to our work in that we do not try to find an optimal configuration for a matcher. Most importantly, unlike both described works, we do not require a human-constructed set of reference links to find the most suitable domain filter.

Chapter 3

Approach and Implementation

In this chapter we describe our approach to domain-aware ontology matching. As described in the previous section, research on ontology matching has most commonly focused on string similarity-based matching – that is, comparing strings that describe entities, such as labels, descriptions, etc, in order to find correspondences between entities. However, little is known about how the *domain* of entities can be considered to assist us in disambiguating source entities to target entities, which is what we will focus on for our approach. Additionally, we explore to what extent the generation of a domain-restricted matcher can be automated through the exploitation of existing structural information of target entities, and see whether the domain restriction can be optimized.

What we essentially want to find out is how the domain of source entities can be mapped effectively to a domain of classes and categories in DBpedia, in such a way that the target domain effectively encapsulates the source domain (see figure 3.1). We can then use this target domain as a filter for the traditional matching process done by (complex) string comparison. The reason we need to derive a union of classes rather than a single class is that there is usually no "perfect" mapping to a single target class. This is because the classes and categories contained in DBpedia are either user-defined (in the case of Wikipedia infobox types, categories and Freebase classes) or automatically generated (in the case of YAGO classes, which are derived from the Wikipedia category system using WordNet [49]). Hence, given the large number of resources in DBpedia, not all resources have proper classes or categories assigned to them, so some redundancy in the domain encapsulation helps deal with this kind of messy data.

In this chapter we first describe DBpedia Spotlight, a text annotation tool that we employ throughout the work to assist us with the matching of entities to DBpedia. The full approach – that is, domain derivation followed by the actual matching – can be broken down into a number of steps that are sequentially performed. Each subsequent section will describe the steps of the approach in detail by hand of a running example to make the explanations easier to understand. The final section will shed some light on the physical implementation of the approach, and we end the chapter with a conclusion.

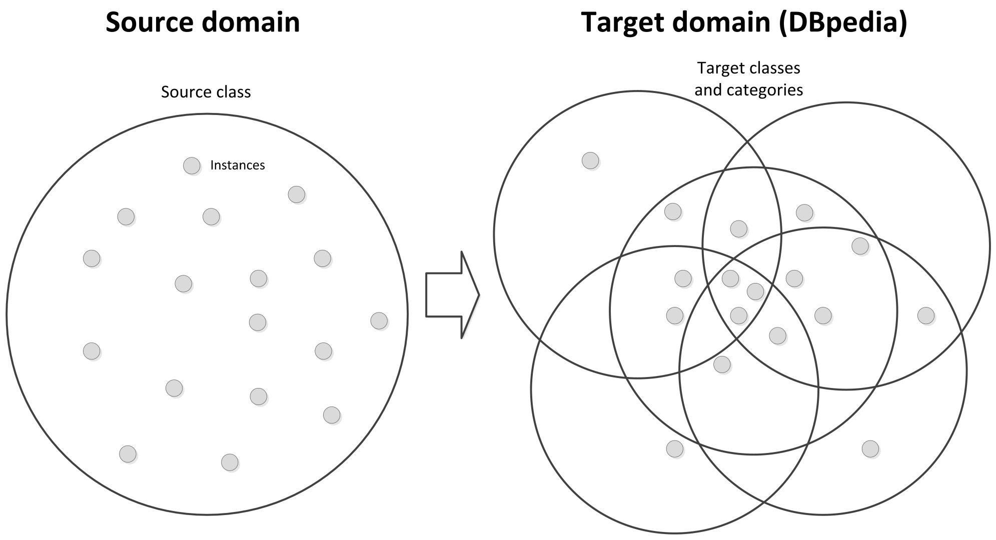


Figure 3.1: Mapping from source to target domain.

3.1 DBpedia Spotlight

As a starting point for the actual matching work we take DBpedia Spotlight [35], a powerful, off-the-shelf tool for matching textual resources to DBpedia. DBpedia Spotlight has been shown to be able to compete with established annotation systems while remaining largely configurable [35]. Its configurability allows us to include various forms of domain knowledge, and test the effect on the resulting matches.

Spotlight works by first finding *surface forms* in the text that could be mentions of DBpedia resources (the "spotting" function), then disambiguating these surface forms to link to the right DBpedia resources based on context similarity measures (the "disambiguation" function). Its results can be directed towards high precision or high recall by setting two parameters: a "support" threshold for minimum popularity of the associated Wikipedia page (i.e. the number of inlinks to this page from other Wikipedia pages) and a "confidence" threshold for minimum similarity between source text and context associated with DBpedia surface forms. The confidence of a match also takes into account the difference in similarity score with the second-most similar concept: the confidence score gets an increasing downward bias as this difference gets smaller, so that ambiguous resources are assigned a lower confidence value (i.e. we cannot be fully sure that this resource is the correct one since there are some very similar resources to this one) and unambiguous resources get higher confidence (i.e. we can be pretty sure that the resource we end up with is correct since there is no other resource like it). The confidence score has been normalized to a range of 0..1. In addition, Spotlight's "black- and whitelists" allow one to filter the results to exclude/include only members of certain classes and categories that correspond to the domain of the source text. It also allows filtering based on SPARQL queries.

As mentioned, the tool performs disambiguation of possible resource mentions based on surface forms that may point to these resources. These surface forms are mined from Wikipedia in a pre-processing stage. For each resource, the following surface forms and context information are collected:

- the resource labels, i.e. the Wikipedia page titles, with the entire page as context
- redirect and disambiguation labels, i.e. titles of pages that redirect or disambiguate to the resource
- anchor texts of hyperlinks pointing to this resource from other Wikipedia pages, with the sentence the link occurs in as context
- anchor texts of hyperlinks pointing to this resource from the Web, with the sentence the link occurs in as context

The various types of context gathered for each resource are combined and stored along with the resource URI and its associated surface forms. When trying to disambiguate a surface form to a resource, all resources associated with this particular surface form are gathered and ranked according to the similarity between the context of the surface form (if available) and the context belonging to each resource.

For our work we do not employ the full range of Spotlight's features – the tool mostly specializes in the annotation of free text, while we deal with a very specific case of text annotation, namely that of entity labels. In the majority of cases it suffices to simply try to disambiguate this full label to find a corresponding DBpedia resource. However, entity labels may be messy: they might contain typos, irrelevant information or a single label might even contain multiple entity references. In order to address these cases we can employ Spotlight's annotation function to increase the number of matches found.

3.2 Running example

To illustrate our approach more clearly, we take as example the EventMedia Last.fm artist dataset described in section 2.4.1. This running example serves two purposes: firstly as a basis for evaluation of key choices that we make in constructing our approach, and secondly to more clearly explain and demonstrate the approach. The dataset is in RDF format, and consists of around 50,000 performing artists mined from the website Last.fm. Each artist instance is comprised of a name and most of the time a description that describes this artist. For this dataset we have manually created a reference alignment consisting of 1000 artist entities and their corresponding DBpedia resources. We found that matches to DBpedia can be discovered for roughly 36% of the artists.

3.3 Domain-aware approach

This section will detail the specifics of our domain-aware approach to ontology matching. Figure 3.2 shows a high-level schematic of the pipeline employed. The approach is fed input in the form of an ontology (see section 3.3.1), then starts out with a *bootstrapping phase*. Here, a sample of high-confidence matches to DBpedia resources is generated, from which classes/categories are collected and a bootstrap linkset is created. The bootstrapping phase is explained in further detail in sections 3.3.2 and 3.3.3.

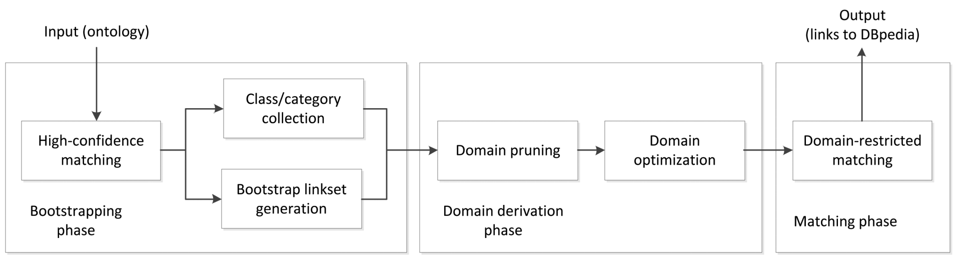


Figure 3.2: High-level pipeline view for the domain-aware approach.

Next is the *domain derivation phase*, where the collection of classes and categories is pruned and optimized. This is expanded upon in sections 3.3.4, 3.3.5 and 3.3.6. Lastly, the actual (domain-restricted) matching is performed, in the *matching phase*, to obtain links to DBpedia as output. This is described in section 3.3.7.

3.3.1 Input

As input we take an ontology or dataset, along with the class of the entities we want to match, and the properties pointing to the label(s) and context of each entity. The class is required so that we know where the entities are located, in case there are multiple classes within a dataset. All entities that belong to this class or its sub-classes will be selected for processing. There can be more than one input label, in which case we distinguish between a *preferred* label and *alternate* labels. A preferred label is required; there may be 0 or more alternate labels. Given the example of the Last.fm dataset, we want to match artists, so we give as input the EventMedia RDF dataset, as entity class `foaf:Agent`, and as label and context properties `rdfs:label` and `dc:description` respectively.

3.3.2 Bootstrapping

Initially, we assume to have no knowledge about our input dataset other than the class type of the entities that we want to match and the location of labels describing each entity and descriptions providing extra context information about this entity. Note however that even if we do already know the domain of our source dataset, it would still not be obvious to which DBpedia domain (i.e. collection of classes and categories) it would map.

The selection of the domain filter is bootstrapped by first attempting to match instances to DBpedia resources without any knowledge of the domain, to obtain an initial linkset from which we want to derive useful information. Ideally, we want to maximize precision for our generated links while still obtaining a sufficiently large sample to make meaningful generalizations for the full dataset. This is accomplished by applying DBpedia Spotlight without any domain restriction and with parameters set towards high precision, thereby relying solely on the similarity of contexts between source and target entities for disambiguation. Entities to match are selected at random from the source dataset. If there are source entities that do not contain context information, then these are ignored in this step. Further, we only check the preferred entity labels and ignore any alternate labels provided. In this way, we can generate an initial set of high-confidence links from our data to DBpedia resources. This set of links is then stored to later be used for two separate purposes:

1. the collection of classes and categories for the derivation of a domain (section 3.3.3)
2. the optimization of the target domain filter (section 3.3.5)

For the first goal, we want a linkset that is large enough to make accurate predictions about the domain. For the second goal, we have a choice of either storing only *positive matches* or a mix of positive and *negative matches* for our initial set of links.

Definition 7 (Positive match)**.** *A correspondence mapping from a source entity to a target DBpedia resource. A single source entity may contain multiple positive matches to (distinct) DBpedia resources. Formally:*

$$\begin{aligned} m_p = \{\langle e_s, Res_t\rangle : e_s \textit{ is a source entity,} \\ Res_t \textit{ is the set of target resources, } Res_t \neq \emptyset\} \end{aligned} \tag{3.1}$$

Definition 8 (Negative match)**.** *A null-mapping for a source entity, i.e. this entity is determined to have no corresponding DBpedia resource. That is:*

$$m_n = \{\langle e_s, \emptyset\rangle : e_s \textit{ is a source entity.}\} \tag{3.2}$$

Subsequently, we can define a *bootstrap linkset* as follows.

Definition 9 (Bootstrap linkset)**.** *A* bootstrap linkset R' *is the set of positive and negative matches obtained by bootstrapping the source dataset, i.e.,*

$$R' = M_{bootstrap} = P_{bootstrap} \cup N_{bootstrap}, \tag{3.3}$$

where $P_{bootstrap} = \{m_{p,i} : 1 \leq i \leq n\}$, $N_{bootstrap} = \{m_{n,j} : 1 \leq j \leq m\}$, $n+m = |R'|$. *Furthermore, for the purposes of this experiment, the bootstrap linkset should be fully disjoint with the real reference alignment R (definition 3):*

$$R \cap R' = \emptyset. \tag{3.4}$$

The variable name R' was chosen since the bootstrap linkset can essentially be regarded as a *virtual* reference alignment. Note that for negative matches, surface forms that do not have corresponding candidates in DBpedia Spotlight's candidate index are left out of consideration, as these matches will be negative irrespective of the domain filter, and therefore uninformative as reference matches. Ergo, only candidates that are found but filtered out by the context similarity threshold are considered negative matches.

We experiment with both positive and negative matches in the bootstrap linkset, and evaluate their relative effectiveness in section 3.3.5.1. For evaluation purposes, we stick with $|R'| = |R| = 1000$ for now. We test how results change for different sizes of R' at a later stage.

3.3.2.1 Parameter settings

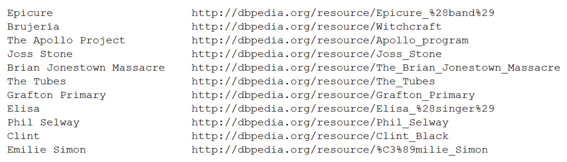
The exact value to choose for “confidence,” the threshold parameter for context similarity in Spotlight, is somewhat dependent on the nature of our domain – in the case of a very specialized domain, the proportion of entities in the source dataset that have a corresponding resource in DBpedia may not be high. If we then set the confidence threshold too high, we may end up with too few matches to obtain sufficiently accurate domain information, and a reference set with too little (positive) matches. On the other hand, setting the confidence threshold too low will reduce the quality of matches, and subsequently the quality of domain information. If we are dealing with a more general domain with many corresponding resources in DBpedia, however, we can set this threshold a little higher, thus obtaining better quality domain information – which is indeed essential in this scenario, since a more generic domain also translates to higher entity disambiguation difficulty. Additionally, there is an inherent duality between quality of matches and computational load – the higher the threshold, the more potential matches we discard and the more surface forms we need to process until we have a linkset that fits the size requirement. For the purposes of this work, however, we do not consider computation speed to be an important factor, especially for the bootstrapping phase, which only has to be performed once.

We choose to start with a default confidence threshold value, then make upward or downward adjustments to this threshold if needed by counting the number of resulting matches over the full source dataset. If we collect only positive matches, there obviously need to be enough result links for the chosen linkset size requirement. If we collect both positive and negative matches, we require at least a sizeable portion of both types of matches in the linkset. Since it is not immediately clear which ratio of positive to negative matches is ideal, we test two cases. In total, we have the following options for the bootstrap linkset:

- 100% of the linkset consists of positive matches. For example, given $|R'| = 1000$, then $|P_{bootstrap}| = 1000$ and $|N_{bootstrap}| = 0$.
- 75% of the linkset consists of positive matches, and 25% negative. Given $|R'| = 1000$, then $|P_{bootstrap}| = 750$ and $|N_{bootstrap}| = 250$.
- 50% of the linkset consists of positive matches, and 50% negative.

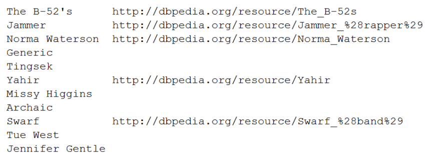
The positive and negative matches m_p and m_n are gathered for two *separate* confidence thresholds: to be maximally confident in positive matches, the threshold should be high, i.e. high context similarity and/or low ambiguity are demanded; to be maximally confident in negative matches, the threshold should be low, i.e. low context similarity and/or high ambiguity are demanded. We test both thresholds until we obtain boundary values where we can obtain just enough matches for each type.

The reason why there is a bias for less negative matches than positive is that it is inherently much harder to determine when a match *should not* be made than when it *should* be made, and therefore these values are the most intuitive. For convenience, we will henceforth refer to the three different linksets as $R'_{1000/0}$, $R'_{750/250}$ and $R'_{500/500}$, where the subscripts denote the $|P_{bootstrap}|/|N_{bootstrap}|$-ratio.



Epicure	http://dbpedia.org/resource/Epicure_%28band%29
Brujería	http://dbpedia.org/resource/Witchcraft
The Apollo Project	http://dbpedia.org/resource/Apollo_program
Joss Stone	http://dbpedia.org/resource/Joss_Stone
Brian Jonestown Massacre	http://dbpedia.org/resource/The_Brian_Jonestown_Massacre
The Tubes	http://dbpedia.org/resource/The_Tubes
Grafton Primary	http://dbpedia.org/resource/Grafton_Primary
Elisa	http://dbpedia.org/resource/Elisa_%28singer%29
Phil Selway	http://dbpedia.org/resource/Phil_Selway
Clint	http://dbpedia.org/resource/Clint_Black
Emilie Simon	http://dbpedia.org/resource/%C3%89milie_Simon

(a) A linkset description with only positive matches.



The B-52's	http://dbpedia.org/resource/The_B-52s
Jammer	http://dbpedia.org/resource/Jammer_%28rapper%29
Norma Waterson	http://dbpedia.org/resource/Norma_Waterson
Generic	
Tingsek	
Yahir	http://dbpedia.org/resource/Yahir
Missy Higgins	
Archaic	
Swarf	http://dbpedia.org/resource/Swarf_%28band%29
Tue West	
Jennifer Gentle	

(b) A linkset description with both positive and negative matches.

Figure 3.3: Linkset description excerpts: each match m is represented as a $\langle label, URI \rangle$ pair, where $label$ is the preferred label of the source entity and URI is a set of 0 or more DBpedia resource URIs. The descriptions are stored in a simple tab-separated value (TSV) format.

3.3.2.2 Last.fm use case

We generate the described linkset types for the use case of Last.fm artists. We ensure that these sets are disjoint with the manually created reference alignment. Due to the reliance on context similarity for matching, artist instances that do not include a description are ignored in order to improve the quality of the linksets.

For $R'_{1000/0}$, we test confidence threshold values c down from $c = 1.0$ in steps of 0.1. $c = 0.6$ is the first value to result in a linkset of 1000 matches. While this value may differ given a different dataset, and may be set higher when considering smaller bootstrap linksets, it should suffice as an initial, "rough" default value. We therefore take the value 0.6 as the default bootstrap confidence threshold for positive matches, used as a starting point for finding a better boundary value, for the remainder of the approach.

For $R'_{750/250}$ and $R'_{500/500}$, we test confidence threshold values for negative matches up from $c = 0.1$ in steps of 0.1. Here, $c = 0.1$ already results in the 250 m_n needed for the first set, so this is what is chosen as default. $c = 0.2$ gives the 500 m_n needed for the second set. For the positive matches, we set c at 0.6 and 0.8 respectively. See figure 3.3 for excerpts of the resulting linkset descriptions.

3.3.3 Class/category collection

From the DBpedia resources that were obtained in the bootstrap step – that is, the positive matches contained in the bootstrap linkset – we gather the associated classes and categories so that we can later use these as a domain knowledge filter to find better quality matches. The following four types of classes are gathered:

1. *DBpedia classes.* These are derived from the manually assigned infobox types on Wikipedia pages. The DBpedia classes are ordered in a shallow, strict hierarchy (i.e. each class has no more than one super-class), so we also gather all super-classes up to the root of the hierarchy.

2. *Wikipedia categories.* These are the categories manually assigned to Wikipedia pages. They are ordered in a broad and non-strict hierarchy (i.e. categories can have multiple super-categories). Therefore we gather only up to 4 ancestors of each, as due to the size and messy structure of the category hierarchy the set would quickly become too large and broad to be useful.

Table 3.1: An overview of the properties of the different types of classes/categories that are gathered and used as a domain knowledge filter.

Type	**Source**	**Creation method**	**Size**	**Hier. type**	**Hierarchy depth**	**Ancestors gathered**
DBpedia classes	Infobox types	User-defined	>320	Strict	Shallow (max 7)	All
Wikipedia categories	Wikipedia	User-defined	>700,000	Non-strict	Deep	4
YAGO classes	Categories + WordNet	Auto-generated	>100,000	Near-strict	Deep	All
Freebase classes	Freebase classes	User-defined	>1000	Strict	2 levels	All

3. *YAGO classes.* These are automatically derived from the Wikipedia category system using WordNet [36][49]. The accuracy of the derivation employed is claimed to be around 95%. YAGO classes are ordered in a deep, near-strict hierarchy (a select few classes have multiple super-classes), so we gather all super-classes up to the root of the hierarchy.

4. *Freebase classes.* These are taken from the Freebase concept associated with a DBpedia resource (if available). Links between DBpedia and Freebase resources have been derived through the common link of Wikipedia – if a Freebase concept and a DBpedia resource link to the same Wikipedia page, they are considered identical. Freebase classes are ordered in a shallow, strict hierarchy of only two levels with no single root node, so all we gather for each concept is the class and its super-class.

The information is summarized in table 3.1. A *domain* is then formally defined as:

Definition 10 (Domain)**.** *A domain D is the set of DBpedia classes, Wikipedia categories, YAGO classes and Freebase classes c_i, and includes the number of occurrence $|c_i|$ of each, i.e.*

$$D = \{\langle c_i, |c_i| \rangle : c \textit{ is a class/category, } 1 \leq i \leq |c|\} \tag{3.5}$$

As per definition 10, each unique class or category is stored along with their number of occurrence. Figure 3.4 shows a section of the YAGO class hierarchy based on our example dataset, with the YAGO classes and their occurrence numbers that we collected from a subset of 100 of the DBpedia resources that were matched in the bootstrap linkset from the previous section. We use this collection as a basis for the derivation of an effective domain filter. All subsequent experiments will assume a class/category collection size of 100. We choose this value to ensure that it can be kept constant when comparing bootstrap linksets of different sizes at a later stage.

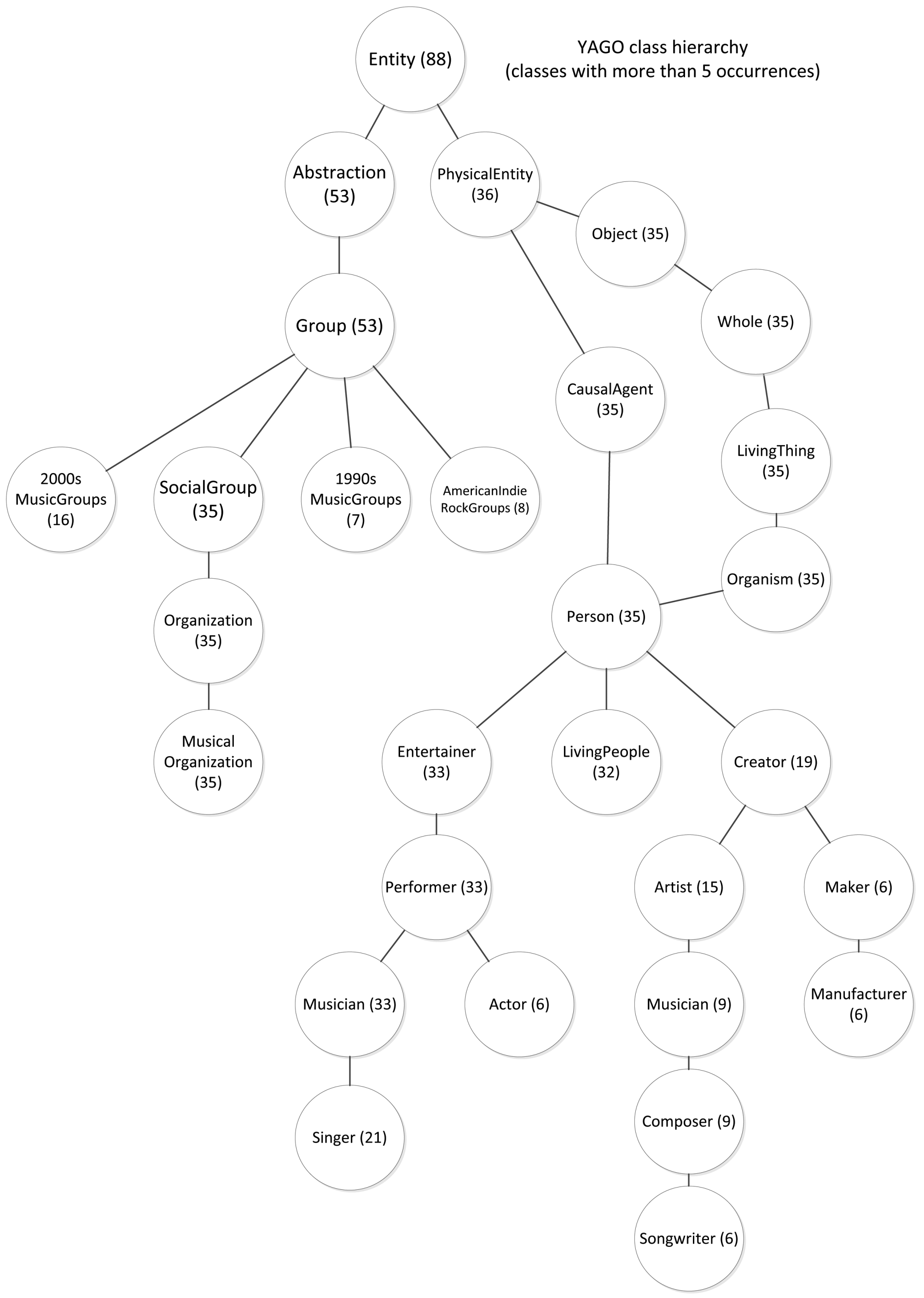


Figure 3.4: The top of the YAGO class hierarchy when collecting classes/categories from 100 DBpedia resources. Values within parentheses are the occurrence numbers.

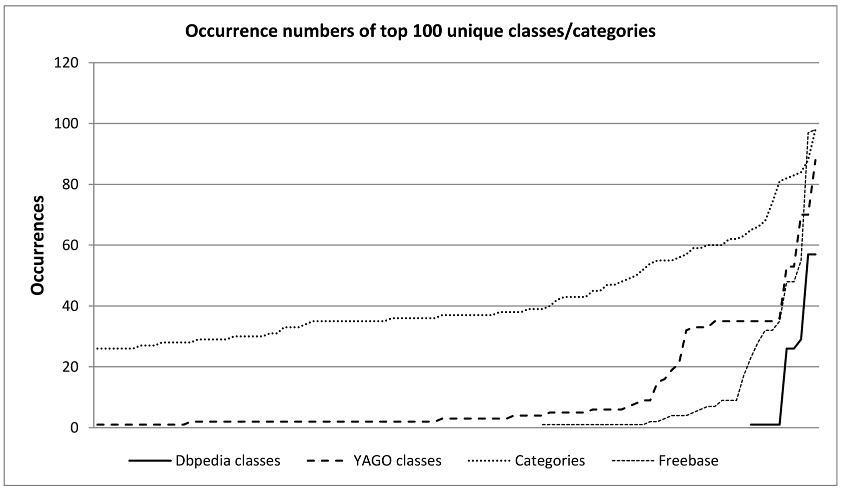


Figure 3.5: Numbers of occurrence for the top 100 unique classes/categories collected from 100 DBpedia resources, sorted by size on the horizontal axis. In total, there are 200 DBpedia class occurrences (10 unique), 1248 YAGO class occurrences (374 unique), 14,219 category occurrences (3901 unique) and 599 Freebase class occurrences (39 unique).

3.3.4 Domain pruning

A collection of classes and categories can be pruned in order to get a tighter encapsulation for the domain. There is little point in using the collection as-is as a domain filter because DBpedia and YAGO classes are collected up to the root of the hierarchy. This means that root classes Thing and Entity of each respective hierarchy are always included, which cover every single DBpedia resource. Therefore, we need to discard classes that are too broad and cover far more entities than what we want it to do. Furthermore, classes that only a occur once or twice over the entire class/category set are typically better left out, as these are likely to either be outliers to the domain of interest. Even if not, they are too narrow to play any role.

Figure 3.5 shows an example of the numbers of occurrence for the top 100 most often occurring unique classes and categories when we collect them for 100 DBpedia resources. It can be seen that most of the occurrences are contained in the top 20 or so unique classes/categories, and that tails can be extremely long: for example, there are 3901 unique categories in total. Since we want classes that all entities have in common, there is relatively little harm in dropping these long tails.

We develop two *pruning strategies* that can be applied to somewhat limit the initial domain, all the while taking care not to discard potentially relevant domain information.

3.3.4.1 Low-occurrence domain pruning (LDP)

We can get rid of classes/categories that occur too little as follows. We filter out all DBpedia and YAGO classes that occur less than $r_d\%$ resp. less than $r_y\%$ of the total number of classes found. The intuition behind this is that, by the law of large numbers, a class that only occurs a small number of times has the highest probability of being a mismatch, and if it is not, it is likely that this class is too specific to be of any significance in the final domain encapsulation, and can therefore be safely pruned. Furthermore, since we gather all descendants up to the root of the hierarchy, general (super-)classes will occur more frequently than specific (sub-)classes. Therefore, the higher the values of r_d and r_y, the more general the list of classes will be. For categories, filtering by a percentage of total occurrences makes less sense since we do not gather super-categories up to the root. Since we only collect them within a specified semantic distance, the very categories with the most occurrences should be a relatively accurate representation of our domain. Therefore we simply select the top t_c categories that occur the most. For Freebase classes, there are only two levels in the hierarchy, hence this effect does not occur here either: we select simply the top t_f classes.

As default parameter values for this method, we choose $r_d = r_y = 1\%$, $t_c = 10$ and $t_f = 6$ to ensure that a fairly broad selection of classes/categories is obtained, and that we do not cut out potentially important information. For our example, we also stick with this default. For the example YAGO classes, $r_y = 1\%$ leads to a cutoff of 14. For the classes displayed in figure 3.4, Actor, Songwriter, Composer, Musician (1 of 2), Manufacturer and Maker are discarded.

3.3.4.2 High-generality domain pruning (HDP)

Even after applying low-occurrence domain pruning, the current domain will most likely not yield a satisfactory result when used as a matching filter, primarily because it contains very general classes from high up in the hierarchies of the DBpedia and YAGO class trees, which could potentially cover each and every entity contained in DBpedia. In other words, a strategy is needed to eliminate excessive redundancy in the list of classes and categories, i.e. to avoid including a super-class plus all of its sub-classes. We can address this by filtering out all super-classes where the sum of the numbers of occurrence of their *direct* sub-classes is greater than $p\%$ of the number of occurrence of the super-class. In other words, for every class $Super$, if

$$\sum_{i=1}^{n} |Sub_i| > \frac{p}{100}|Super|, \tag{3.6}$$

where n is the number of direct sub-classes Sub_i of $Super$, then $Super$ is removed from the collection. The same procedure is followed for categories. This procedure is dependent on the values chosen for r_d, r_y, t_c and t_f in the low-occurrence class pruning step. As default we choose to take $p = 80\%$. See figure 3.6 for an illustration of this pruning step based on the example with this default value. As can be seen, most generic classes such as “Entity” and “Group” are removed and by intuition we get a relatively good representation of the “musical artists” domain. However, some fairly generic classes, such as “LivingPeople,” still remain, which suggests that we need some extra measures in order to further refine, or *optimize*, the domain encapsulation.

3.3.5 Domain optimization: testing performance

Once we have gathered a full domain collection of classes and categories, we can apply several methods to further tighten the coverage of the domain filter for higher quality matching. One of such methods was illustrated in the previous section. Ideally, we

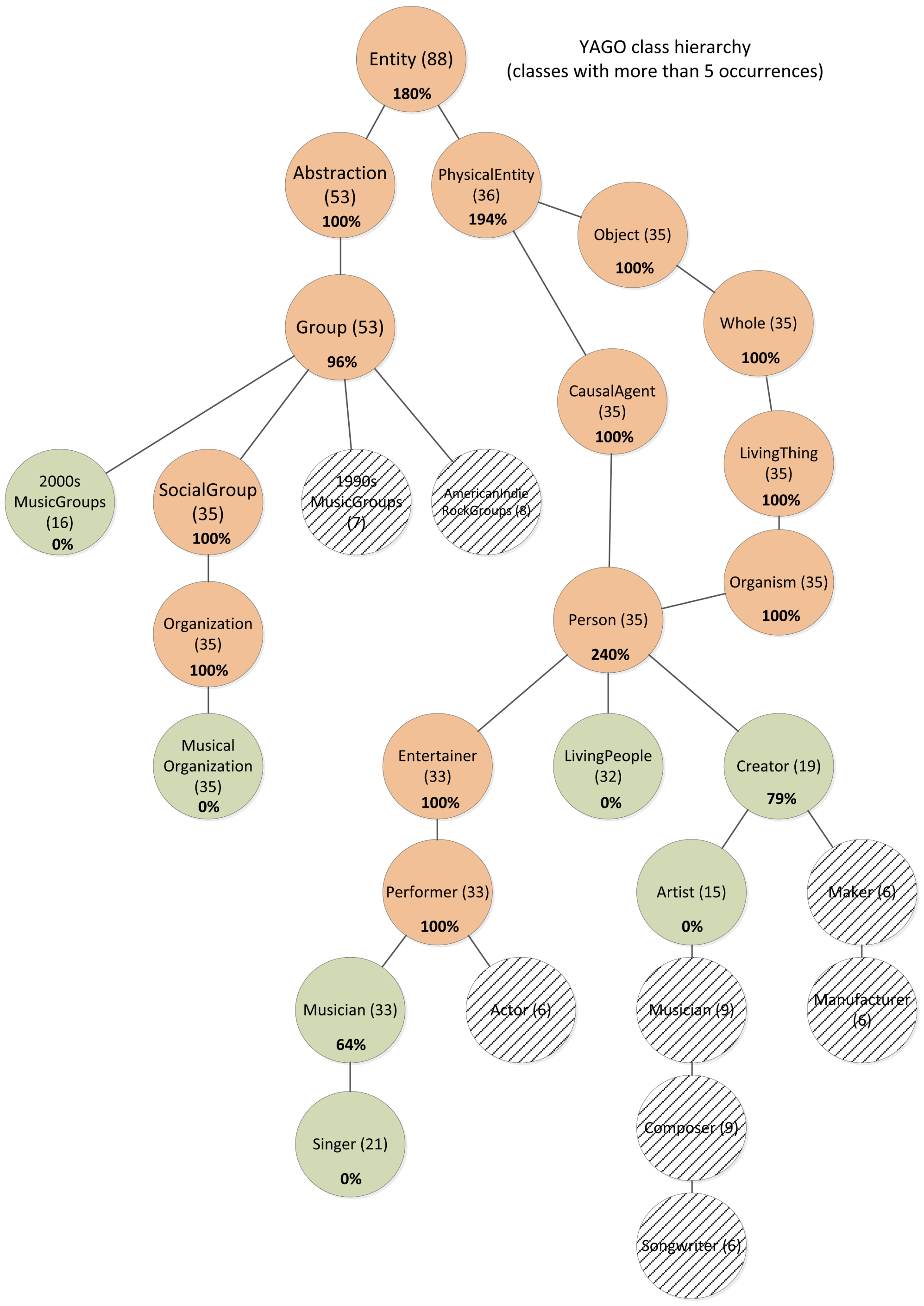


Figure 3.6: The top of the YAGO class hierarchy after pruning with $p = 80\%$. The percentages are the proportions of direct sub-class occurrences to this class' occurrences. Classes striped out were already discarded in the LDP step; classes marked red are discarded in the HDP step. Classes marked green represent the final domain.

want to filter out any remaining classes that are too general for the domain of interest, but at the same time, the classes we want to keep must still be general enough to fully cover the domain of interest. That is, they must not be overfitted to the set of resources from which the domain information was originally obtained, i.e. the bootstrap linkset.

In other words, we are faced with a non-trivial optimization problem of selecting the optimal collection of classes that most effectively captures the domain of interest. One approach is to prune the domain selection based on structural information, as shown in the previous section. Still, we cannot be sure that this will provide the best result; ideally, we want to be able to *test* a method's performance and *optimize* the solution space of possible domain selections. We devise several methods to solve this problem, described in the following subsections.

3.3.5.1 Testing performance on a bootstrap linkset

The set of links created in the bootstrap step (see section 3.3.2) could theoretically be used as a (virtual) reference alignment to test a domain restriction on, provided that there exists a correlation between performance according to this automatically generated bootstrap linkset R' and performance on the actual dataset (i.e. according to the manually constructed reference alignment R). Since the bootstrap linksets were created with high confidence settings, we can assume a large percentage of the matches made to be correct, but just this assumption is not sufficient evidence to imply a correlation.

Because we relied on context similarity, and only kept those links above a certain (very high) similarity threshold for positive matches, and below a very low threshold for negative matches, we end up with only those links that are the most obvious and least ambiguous. However, for traditional automated link discovery approaches, the most interesting and useful example matches are in fact the ones that are the most ambiguous and challenging to derive [37], which are in turn difficult or even impossible to discover automatically. It is for that reason that virtually all existing state-of-the-art approaches require at least a handful of example links provided by a human domain expert in order to compute or learn an effective matcher. We observe that in our case, however, the domain filter we are trying to optimize is used to *restrict* matches, and is fully independent from the settings used to *create* the matches. This implies that for the optimization of a domain filter with an automatically generated example linkset, the ambiguity and difficulty of the automatically derived matches is irrelevant – only the quality of the matches will have influence on the result.

To test virtual performance of a domain filter, we can generate an *alignment instance* over the entities contained in the bootstrap linkset that is restricted by this domain.

Definition 11 (Alignment instance)**.** *An* alignment instance A_X *over a linkset X is a set of matches generated from all entities contained in a linkset X to target DBpedia resources, i.e.,*

$$\begin{aligned} A_X = \{\langle e_i, Res_t\rangle : & \ e_i \textit{ is an entity in linkset } X, \\ & Res_t \textit{ is the set of target resources, } 1 \leq i \leq |X|\}. \end{aligned} \tag{3.7}$$

An alignment instance that is limited to a domain filter is then defined as follows.

Definition 12 (Domain-restricted alignment instance)**.** *A* domain-restricted alignment instance $A_{X,D}$ *over a linkset X is a set of matches generated from all entities contained in a linkset X to target DBpedia resources, provided that each target resource is contained in domain D (i.e. has a class/category included in D). That is,*

$$\begin{aligned} A_{X,D} = \{\langle e_i, Res_t \rangle : & \; e_i \text{ is an entity in linkset } X, \\ & Res_t \text{ is the set of target resources, } 1 \leq i \leq |X|, \; Res_t \in D\}. \end{aligned} \tag{3.8}$$

Subsequently, we can define the F_1-score for an alignment instance $A_{R'}$ over the bootstrap linkset as follows.

Definition 13 (Virtual F_1-score)**.** *Given an alignment instance $A_{R'}$ of the entities of a bootstrap linkset R', the* virtual F_1-score f_1 *for $A_{R'}$ is the F_1-score calculated by using R' as reference alignment for $A_{R'}$.*

As outlined in section 3.3.2, for the automatic evaluation of matches by means of virtual F-scores we can use either a bootstrap linkset consisting of solely positive matches, or a linkset with both positive and negative matches, i.e. entities for which no matching resource was found, and for which we therefore do not expect a match in the result. Recall that we ended up with three types of bootstrap linksets: $R'_{1000/0}$, $R'_{750/250}$ and $R'_{500/500}$.

For $R'_{1000/0}$, we consider only positive matches, so when generating the alignment instances $A_{R'}$ we want to retrieve as many results as possible (i.e. high recall), so that the probability to obtain the wrong links given a domain restriction that is too broad is high. This makes it easier to filter out classes that are too broad for the desired domain. To realize this, per entity we can choose to consider not just the first candidate resource found that fits the domain filter, but also all returned candidates. To further strengthen this effect, we can choose to ignore the context information associated to entities: this will yield a different ranked list of candidate resources per entity. Note that if context *is* included, the ranking of candidates per entity for $A_{R'}$ would be the same as the ranking of candidates per entity for $M_{bootstrap}$, since the matching approach is kept the same.

For $R'_{750/250}$ and $R'_{500/500}$, we consider both positive and negative matches, and can therefore judge based on false positive matches in $A_{R'}$ for entities that did not have matching DBpedia resources in R' as well. Since it is now more similar to the original reference alignment R, it seems more intuitive to keep the matching approach to obtain $A_{R'}$ similar to the one employed to obtain $M_{bootstrap}$ (i.e. consider context and only the first matching candidate).

As it is not immediately evident which is the better alternative, we test all possible approaches and evaluate which is the most suitable. We thus end up with four different possible matching approaches for three different bootstrap linkset types. All 12 methods are tested and compared with each other in order to determine the most effective method.

3.3.5.2 Correlation with performance on the reference alignment

The key to having the bootstrap linkset R' be effective for optimization purposes is that there exists a strong correlation between R' and the actual reference alignment R,

Table 3.2: Twelve domain selection strategies to obtain maximum variability for an accurate correlation evaluation. Shown are the LDP parameters used and whether or not HDP was applied. Three random selections are taken from the resulting domain selection for each strategy.

Pruning strategy	r_d	r_y	t_c	t_f	*HDP* applied
1	1%	1%	10	6	No
2	1%	1%	10	6	Yes
3	2%	2%	10	6	No
4	2%	2%	10	6	Yes
5	1%	1%	5	3	No
6	1%	1%	5	3	Yes
7	2%	2%	5	3	No
8	2%	2%	5	3	Yes
9	1%	1%	0	0	No
10	1%	1%	0	0	Yes
11	2%	2%	0	0	No
12	2%	2%	0	0	Yes

since good performance of a matcher according to R' needs to translate to good performance of the same matcher on the actual dataset. We evaluate the different bootstrap linkset types and matching approaches defined in the previous section and determine the correlations between performance on R' and R; the method providing the highest correlation is chosen for the optimization part of our approach. The experiments are first executed on the Last.fm Artist use case, with bootstrap linkset sizes of 1000 and a domain collection size of 100, as described in the examples. We then do the same for the UMLS use case, to test the robustness of this correlation.

We evaluate by comparing f_1-scores obtained by comparison with R' and F_1-scores obtained by comparison with R, for random domain filters. To be able to accurately determine a correlation, a significant variability in matching scores is required in order to increase the statistical significance of the result: after all, it needs to be shown that if scores are low on the bootstrap set, they should be low on the real dataset, and if they are high on the bootstrap set, they should be high on the real dataset as well. Therefore we want to obtain maximum variability in the random domain selections, and by extension in the resulting F-scores. To accomplish this, low-occurrence domain pruning (LDP, see 3.3.4.1) is first applied on the initial domain collection, with six different parameter settings. Subsequently, for each of the resulting trimmed domain selections we generate one additional, more compact selection by applying the high-generality domain pruning (HDP) method described in section 3.3.4.2. In total, we end up with 12 different domain selection strategies (see table 3.2). For each domain selection obtained by these strategies, we generate three random class/category selections by randomly including/excluding each class/category with a probability of $\frac{1}{2}$, to end up with 36 final domain selections per run.

The matching approach used to obtain F_1-scores according to the actual reference

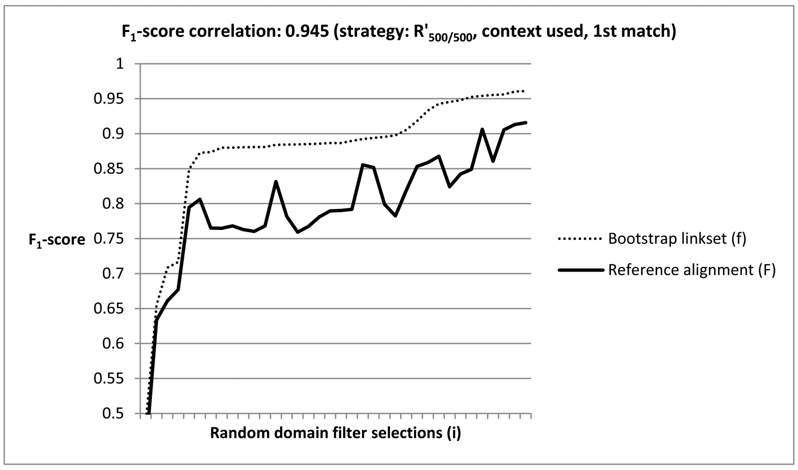


Figure 3.7: Correlation between the bootstrap linkset and the reference alignment for an example run (Last.fm use case). Values have been sorted by $f_{1,i}$ in ascending order to show the correlation more clearly.

alignment R is kept constant throughout: we attempt to disambiguate the full labels of the source entities of R by setting both Spotlight "confidence" and "support" thresholds to 0, and considering the top-2 returned ranked candidates. This way, it is ensured that matching quality will be maximally dependent on the domain filter.

The calculation of correlation values is done by applying the following algorithm on each of the 36 final domain selections.

1. Perform two matching rounds – once on the entities contained in the bootstrap linkset R' and once on the entities contained in the real reference alignment R.
2. Calculate the F-scores $f_{1,i}$ and $F_{1,i}$ for the resulting matches with R' resp. R, where i is the domain selection index ($i = 1,2,\ldots,36$). If $f_{1,i} = 0$ (this might happen if the random domain selector ends up selecting an empty domain, therefore rejecting every match), ignore this result to avoid skewing the correlation.
3. Calculate the *sample correlation coefficient* r_{fF} by

$$r_{fF} = \frac{\sum_{i=1}^{n} (F_{1,i} - \bar{F_1})(f_{1,i} - \bar{f_1})}{(n-1)s_F s_f} = \frac{\sum_{i=1}^{36} (F_{1,i} - \bar{F_1})(f_{1,i} - \bar{f_1})}{35 s_F s_f} \tag{3.9}$$

   where $\bar{F_1}$ and $\bar{f_1}$ are the sample means of each set of F-score measurements, and s_F and s_f the sample standard deviations of each set of measurements.

The result of an example run of the algorithm is shown in figure 3.7. It can be seen here that for large differences in scores, the measures are well-correlated, but for minor differences there is some fluctuation.

To obtain a final result that is statistically significant, we apply the algorithm 15 times and calculate the sample mean and standard deviation over the resulting set of

Table 3.3: Correlations between f_1 and F_1 for different types of bootstrap linksets with size $|R'|$ of 1000 for the Last.fm artists. Standard deviation s is displayed within parentheses.

		$R'_{1000/0}$	$R'_{750/250}$	$R'_{500/500}$
Context	*Matches*	$\bar{r}_{fF}$ (s)	$\bar{r}_{fF}$ (s)	$\bar{r}_{fF}$ (s)
No	1	0.826 (0.087)	0.811 (0.033)	0.719 (0.044)
No	All	0.687 (0.099)	0.662 (0.094)	0.531 (0.161)
Yes	1	0.454 (0.420)	0.936 (0.040)	0.978 (0.009)
Yes	All	0.739 (0.095)	0.619 (0.101)	0.478 (0.139)

correlation values. The results for each bootstrap linkset type and matching approach are shown in table 3.3.

Table 3.3 shows that for an R' of size 1000 with 500 positive and 500 negative matches, using context and taking the first match down the ranking of candidates that is contained in the domain filter gives the best and most consistent result (an average correlation of 0.978 with standard deviation of 0.009). We will use this version of R' to optimize and test our domain filters in subsequent sections. As an extra test as to the strategy's robustness, we can evaluate the strategy on a different dataset and see how the correlation compares. For this we use our second use case, the UMLS dataset.

3.3.5.3 Correlation for the UMLS dataset

The same steps that were applied to the Last.fm dataset are applied to UMLS entities of class "Pathologic Function." This dataset is of a very different nature than the Last.fm artist dataset – where artist/band names are often common terms, and will thus yield many possible candidate resources in DBpedia, the pathologic terms in the UMLS dataset are very specific and therefore unambiguous, with very few possible candidate resources. This difference makes it especially interesting to see how our approach will behave on this dataset, and whether a correlation can still be derived in the same way.

For bootstrapping, we again consider only those entities that contain context information (e.g. a MeSH [29] description of the entity). Unlike the Last.fm dataset, the UMLS dataset also contains alternative labels for each entity, so we consider these when matching as well. Since we are only interested in testing the best performing bootstrap linkset format, we generate an $R'_{500/500}$ bootstrap linkset. For gathering positive matches $P_{bootstrap}$, we can set Spotlight's confidence threshold to 0.6 (anything higher will yield too few matches). For negative matches $N_{bootstrap}$, 0.3 gives us just enough matches. As before, we collect classes and categories for 100 of the DBpedia resources contained in $P_{bootstrap}$ and apply the algorithm described in section 3.3.5.2 for 15 samples, then take the average correlation and standard deviation over this sample set. This results in $\bar{r}_{fF} = 0.980$ and $s = 0.007$, which is very consistent with the $\bar{r}_{fF} = 0.978$, $s = 0.009$ obtained for the same method with the Last.fm dataset. This shows that $R'_{500/500}$ is effective regardless of the nature of the underlying dataset.

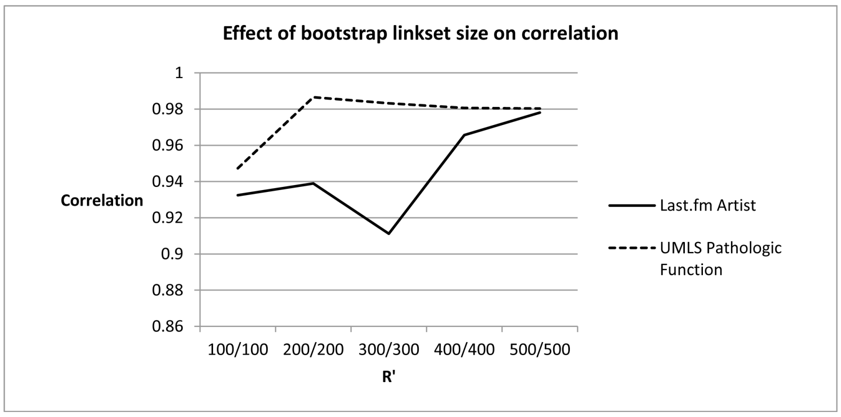


Figure 3.8: Effect on correlation values of Last.fm and UMLS for different bootstrap linkset sizes.

3.3.5.4 Additional constraints on the approach

Minimum required R' size Since it cannot be assumed that the source dataset we are dealing with will always be large enough for us to gather an adequate bootstrap linkset of 1000 matches, it is useful to know to what extent smaller sizes of R' will affect the quality of the overall approach. What is pivotal to good matching performance is that correlation between R' and R remains substantially positive, and standard deviations low. We again test the best-performing approach on the Last.fm and UMLS datasets by taking 15 correlation samples, this time for different sizes of R'.

Figure 3.8 shows the relationship between averaged correlation values for $R'_{500/500}$, $R'_{400/400}$, $R'_{300/300}$, $R'_{200/200}$ and $R'_{100/100}$. A general slight downward trend can be detected as linkset sizes get smaller, with values fluctuating depending on the composition of the linkset, but correlations remain high even for small sizes. So even when collecting many bootstrap links is not an option, we can still presume that the bootstrap linkset will be suitable for testing performance.

Minimum required class/category collection size Now that it is clear which type of bootstrap linkset gives the highest correlation, and how large this set should be, another interesting thing to know is exactly for how many bootstrap links we need to collect classes and categories for to get a statistically reliable result. Since correlations as derived above are completely independent of the size of the original domain selection, we need a different method to find this out. We choose to test this by exploring how different class/category collection sizes influence the final matching results. Since some actual matching strategies need to be devised first, the collection size of 100 is maintained for now, until we can test how different sizes influence the results during the evaluation (chapter 4).

3.3.5.5 Conclusion

We showed that for a bootstrap linkset $R'_{500/500}$, of size 1000 with 500 positive and 500 negative matches, generating alignment instances using context and taking the first match down the ranking of candidates that is contained in the domain filter gives the most consistently correlated result: for the Last.fm use case, we find a correlation between performance on R' and R of 0.978 with a standard deviation of 0.009, and for the UMLS use case a correlation of 0.980 with a standard deviation of 0.007. Taking into account the vastly different natures of the Last.fm and UMLS datasets, we can therefore conclude that this bootstrap linkset can consistently be used as a virtual reference alignment. We will use this bootstrap linkset to test our devised optimization approaches in subsequent sections.

3.3.6 Domain optimization: optimization methods

Given that it is now known which type of bootstrap linkset gives the highest correlation and is thus the most reliable for optimization, the domain selection can be further optimized in a number of different ways. Three methods are tested: a metaheuristic-based approach [30], i.e. generic optimization methods that are known to be able to discover (near-)optimal solutions independent of the nature of the optimization problem itself; and two problem-specific heuristic-based approaches that exploit the structure of our specific problem domain. The performance of each method is compared in the evaluation chapter (4).

3.3.6.1 Metaheuristics

Metaheuristics are a type of stochastic optimization algorithms, and are especially practical for problems for which it is hard to describe how to find an optimal solution, but which allow for easy assessment of the quality of a given solution [30]. They generate or derive solutions in iterations, each time rating each solution according to a quality or fitness function. These solutions are then used as input for the next iteration in attempt to find a solution of superior quality to the previous best. Metaheuristics are usually used to solve optimization problems with very large search spaces where it is not feasible to examine every possible solution. We introduce one of them and later evaluate its performance on our problem.

Genetic algorithm One type of optimization algorithm that is known to perform well on very generic optimization problems is the genetic algorithm. Genetic algorithms emulate the process of evolution as it occurs in nature in order to evolve random starting solutions, or *starting populations*, towards an optimal solution. The only problem-specific requirement for the algorithm is the possibility to evaluate the quality, or *fitness*, of each solution by means of a fitness function – it works otherwise fully independent of the nature of the problem at hand. After generating a random starting population of *chromosomes* – in our case, a bit array where each bit represents whether a class/category from our selection is included or excluded – these chromosomes then go through a number of evolution rounds. A single round of evolution is performed by applying three genetic operators on the chromosomes:

- **Selection**: a portion of the current population of chromosomes is selected for continued breeding. Usually, the fittest portion is selected with high probability. A select few less fit chromosomes are also selected, which may help to prevent the algorithm from getting stuck in local optima.

- **Crossover**: given a pair of chromosomes, genes are crossed-over between each other with a given probability to create new chromosomes.
- **Mutation**: a gene in any chromosome is switched from 0 to 1 with a given probability.

Given these operators, the intermediate solutions are expected to slowly crawl closer towards the optimal solution with every iteration. A downside to this approach is that it is a very time-consuming procedure, since the search space is usually quite broad. No knowledge of the problem is assumed, which means that it is not known in advance where the most likely optimal solution may be, and therefore usually many thousands of evaluations need to be performed. Nevertheless, it is interesting to see how well it performs on our problem in comparison with other approaches.

3.3.6.2 Problem-specific heuristics

Another strategy that can be taken is to exploit any information that is available given the specialized nature of the problem. In our case, we observe that there is information available about the hierarchies of classes/categories, and that next to the measure of quality – the F_1-score – the precision and recall score of an alignment instance are also available. The two methods described below leverage this information to produce algorithms that are less complex and potentially more effective than metaheuristic-based approaches.

Broadness-based domain optimization After applying both low-occurrence domain pruning and high-generality domain pruning on the initial domain selection as described in section 3.3.4, we can try to optimize what remains of the domain selection D by selectively removing those classes that have a high probability of being too broad. One by one, a class or category is removed from the selection, and each time the performance f_1 given an alignment instance $A_{R',D}$ is calculated with the bootstrap linkset R'. If performance improves, this class/category is left out. Else, it is returned to the selection. To maximize the effectiveness of this strategy, the order in which we remove classes/categories and test what remains should preferably be from broad to narrow. After all, removing a narrower class while there is another, broader class in the selection that fully encapsulates this class, would have no effect on the result – even though even this narrower class may still have been too broad to accurately describe the target domain.

Therefore the class/category selection is first sorted by broadness in descending order. Since it is difficult to obtain an exact measure of broadness for a given class, the size of a class, i.e. the number of instances that belong to this class (including its descendants) is taken as an approximation of its broadness. Formally, the algorithm described in Algorithm 1 is applied.

We repeat this algorithm for different settings for r_d, r_y, t_c and t_f of the high-generality domain pruning step, keeping track of the domain selection corresponding to the highest f_1-score seen so far. The resulting domain selection giving the highest F-score according to R' is then our final selection for matching.

Algorithm 1 Broadness-based domain optimization.

```
Require: $D = \{c_n : c \text{ is a class/category}, 1 \leq n \leq \textit{domain size}\}$
  $d_{size} \leftarrow D$.size()
  $D \leftarrow$ sortByBroadnessDesc( $D$ )
  $f_{1,max} \leftarrow$ evaluate( $D$ )
  for $i = 1 \rightarrow d_{size}$ do
    $D$.remove( $i$ )
    $f_{1,current} \leftarrow$ evaluate( $D$ )
    if $f_{1,current} > f_{1,max}$ then
      $f_{1,max} \leftarrow f_{1,current}$
    else
      $D$.return( $i$ )
    end if
  end for
  return $\{D, f_{1,max}\}$
```

Precision-based domain optimization For this approach, we do not apply high-generality domain pruning, but take the result D from the low-occurrence domain pruning step as input. The next step is to test for each individual class/category in D how many entities are contained. This is done by generating alignment instances A_{R',D_1} restricted to single-class domain filters D_1 (i.e. $|D_1| = 1$), and calculating f_1-scores for each. That is, we generate as many A_{R',D_1} as there are classes/categories in D, and for each A_{R',D_1}, the *precision* score is stored. The reason we look only at precision here is that we are now only interested in the quality of matches, not the quantity. Generally, the higher the precision score, the less false positive matches we find and the more relevant the class/category is to our domain of interest. In other words, the higher a precision score, the higher the probability that the class under test is specific to our domain of interest (e.g. restricting to a class "British2000sMusicalGroups" may give a high precision score when the domain of interest is "musical artists"), and vice versa. Therefore we can optimize the domain selection by applying an f_1-based hill-climbing on the list of classes sorted by precision score in descending order. First, we put the class with the highest precision in domain filter D_{test} and calculate f_1-score according to R'. Then we add the second class down the list to D_{test} and calculate f_1 again. We check f_1 again, keeping track of $f_{1,max}$, i.e. the maximum f_1-score seen, and its associated domain filter D_{max}. $f_{1,max}$ and D_{max} are updated every time that $f_{1,current} \geq f_{1,max}$: even if f_{max} remains the same, we prefer the largest domain filter. Once we have gone through the entire list, ending with $D_{test} = D$, the current D_{max} is our final domain filter. See Algorithm 2 for a formal description of the procedure after initial precision values have been gathered.

The domain filter that provided the overall highest f_1-score, D_{max}, is the filter that we take to the matching step.

3.3.7 Matching

Once we have derived a suitable domain filter, the actual matching is performed. As mentioned in section 3.1, we make use of the functionality provided by DBpedia Spot-

Algorithm 2 Precision-based domain optimization.

```
Require: $D = \{c_n : c \text{ is a class/category}, 1 \leq n \leq \textit{domain size}\}$
  $D \leftarrow$ sortByPrecisionDesc( $D$ )
  $d_{size} \leftarrow D$.size()
  $D_{test}$
  $f_{1,max} \leftarrow 0$
  for $i = 1 \rightarrow d_{size}$ do
    $D_{test}$.add( $D$.get( $i$ ) )
    $f_{1,current} \leftarrow$ evaluate( $D_{test}$ )
    if $f_{1,current} \geq f_{1,max}$ then
      $f_{1,max} \leftarrow f_{1,current}$
      $D_{max} \leftarrow D_{test}$
    end if
  end for
  return $\{D_{max}, f_{1,max}\}$
```

light. With the assumption that there is now a filter that tightly encapsulates the domain of the source entities to a domain in DBpedia, the context similarity and ambiguity requirements can be loosened, since disambiguation is now largely performed by this filter. We can divide the matching phase in two separate sub-phases.

Exact match phase First, an *exact match phase* is performed where we attempt to match the full source entity label(s) to a candidate in DBpedia that fits the domain filter. If more than one entity label is provided as input, each label is checked in order until a match is found. The threshold values for "confidence" and "support" are set to 0: now that there is a domain filter, we can be quite confident that a candidate resource mapped to the full label that fits this restriction is the one that we are looking for. Still, not all of the ranked candidate resources returned are considered here; we only look at the top-*k* ranked results, where *k* is a variable of which the optimal value is to be determined. Unfortunately, unlike the domain filter, it is not possible to optimize this variable in an unsupervised way. Therefore we determine an appropriate default value in the evaluation (section 4.4.2).

Partial match phase Optionally, a second matching phase can be applied, this time to try and match entities that are "hidden" within entity labels. For example, for the Last.fm artist dataset, there might be an entity listing several artists in its label that all need to be matched – attempting to match the full label, a possibly arbitrary list of artists, will obviously not yield any candidates. For UMLS, we can consider for example an entity "Acute Bronchitis," where we want just to match "Bronchitis," as with the "Acute" modifier the label would be too specific to find a corresponding resource in DBpedia. This can be addressed by running a second, *partial match phase* on the source entities for which no target resource has been found in the exact match phase. This is done by applying Spotlight's "annotate" function to find surface forms within the labels. Since this method is more error-prone, we can now be much less confident that found target candidates are really the ones that we are looking for. Therefore, the

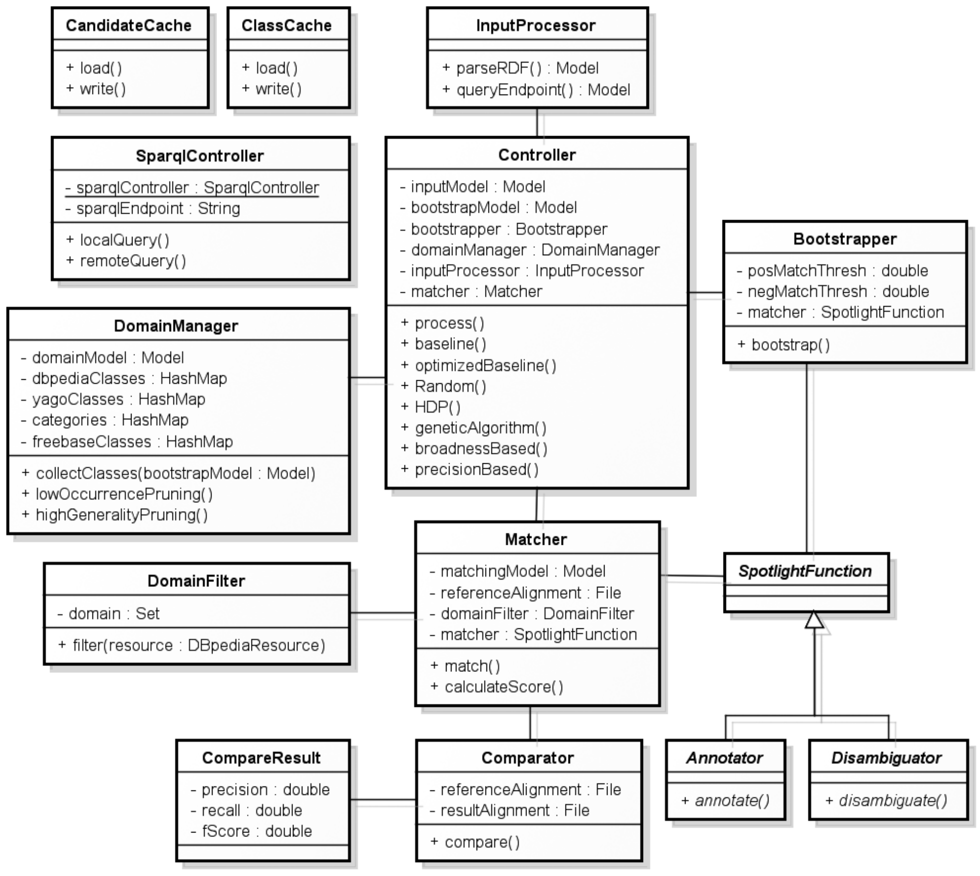


Figure 3.9: An abstracted class diagram for the matching system. Only the essential classes/attributes/methods are shown. Most parameters and return types are omitted.

"confidence" and "support" parameters are set to high values. Which values are ideal (still assuming we are looking to maximize the F_1-score) depends highly on the source dataset. As with the k variable of the exact match phase, it is not possible to determine the best values in a fully unsupervised way. For these we also try to determine reasonable defaults in the evaluation chapter (section 4.4.3).

3.4 Technical implementation

This section will explain in brief the general outline of the system that we developed to implement the described approach. Figure 3.9 presents a simplified class diagram showcasing the fundamental organization of the system. Each component of this diagram will be briefly discussed.

The system has been implemented in Java. For processing RDF, the Jena framework[1] is used. We primarily use Jena models backed by Jena TDB disk-based triple stores for querying and manipulating the data. Jena models are essentially RDF graphs,

[1] http://incubator.apache.org/jena/

providing convenient interfaces for managing and querying the data. Henceforth, each mention of *model* will refer to a Jena RDF model.

The `SparqlController` class exposes a static singleton instance, and is used to process queries to online SPARQL endpoints through their REST/JSON interfaces, or to local models, which are queried using Jena ARQ which supports SPARQL querying.

The `SpotlightFunction` class and its sub-classes are used to interface with the annotate and disambiguate functions of DBpedia Spotlight. Function-specific methods are defined in classes `Annotator` resp. `Disambiguator`. `LocalDisambiguator` and `LocalAnnotator` may be used if Spotlight is installed locally, and call methods from the Spotlight library directly. Otherwise, `RemoteDisambiguator` or `RemoteAnnotator` are used, which make requests to the RESTful DBpedia Spotlight Web demo (these 4 concrete classes are not shown in the diagram for space reasons). `Disambiguator` has a `disambiguate()` method, which returns a ranked list of DBpedia resource candidates given a label and context. For annotation, the `Annotator` class works similarly. The main difference is that it can return multiple DBpedia resources, each with their respective candidates.

The `CandidateCache` and `ClassCache` classes allow for the caching of resources in memory and on disk. Two types of resources are cached: DBpedia resource candidates associated to surface forms, in `CandidateCache`, and the collection of classes and categories associated to a DBpedia resource, in `ClassCache`. This is done to speed up the optimization approaches, which may require evaluations to be repeated a great number of times. Whenever a surface form needs to be disambiguated, for example, it is first checked whether the resource is available in memory; if not, the disk is checked; if it is not found there either, a *Disambiguator* is called to retrieve it from its index. It is then stored in memory and on disk. When the classes/categories of a resource are needed, e.g. to check for domain filter containment, the same steps are performed – if they are neither in memory nor on disk, they are queried from DBpedia (local or remote).

The `Matcher` class is used to generate an alignment instance. It takes a model with entities, a reference alignment in tab-separated value (TSV) format, a domain filter and a Spotlight matcher (`Disambiguator` in most cases) as input. The reference alignment needs to contain exactly those entities contained in the model. For each entity in the model, Spotlight is called with the entity's label and context, which returns candidate DBpedia resources. These are first filtered by confidence/support, then with a `DomainFilter`, which contains a set of classes/categories and checks for a DBpedia resource whether it has any of these classes. If this is the case, the resource is returned to the `Matcher` and subsequently appended to the alignment description. The output of the Spotlight matching process is an alignment description in TSV format. This is compared to the reference alignment with the `Comparator`, which returns a `CompareResult` object containing the precision, recall and F_1-score.

All input to the system is delegated to a configuration file. This file allows the user to set paths used for input and storage and default values for all the parameters that can be adjusted. The `Controller` class regulates the pipeline, and calls components in sequence, starting with the `process()` class. This class performs the following steps:

1. The input to the system is processed by the `InputProcessor`. The input can be stored locally in the form of RDF files or a TDB triple store, or can be located

in an online dataset, in which case a SPARQL endpoint is used. Where to obtain input is defined in the configuration file. Other input settings are the class of entities to gather, the main/alternate label properties and context property. We gather all entities of the defined class and store it and all defined properties in an RDF *input model*, which is stored on disk. This is done so that the input does not need to be reparsed or requeried the second time.

2. Once we have this *input model*, a `Bootstrapper` object is created, which takes as input the input model and a location to store a *bootstrap model*. Bootstrapping settings are read from the configuration file, such as whether to consider context of entities or not, the type of bootstrap linkset to generate and default bootstrap Spotlight thresholds. The `bootstrap()` method iteratively generates the bootstrap model by processing the labels and context of each entity in the input model by feeding them one by one to the `Disambiguator`. It alters the different confidence thresholds for positive and negative matches until it finds the highest resp. lowest possible value given the desired bootstrap linkset size.

3. Once the right bootstrap model has been created, the bootstrap linkset description (i.e. the virtual reference alignment) is generated as a TSV file, containing for each resource the main label with corresponding DBpedia resource URIs.

4. The next step is to gather classes and categories for the prescribed number of resources. This is done in the `DomainManager` class, which takes as input the bootstrap model and a location to store a *domain model*. DBpedia classes and Freebase classes are already contained in Spotlight's candidate index for every DBpedia resource, so these can be gathered immediately. What remains are YAGO classes and Wikipedia categories. These are gathered by querying DBpedia with the following query:

```
DEFINE input:inference
          "http://dbpedia.org/resource/inference/rules/yago#"
SELECT DISTINCT ?classes ?categories ?ancestors WHERE {
    { <%URI%> rdf:type ?classes } UNION
    { <%URI%> dcterms:subject ?categories .
      ?categories skos:broader ?ancestors
          option(TRANSITIVE, T_DISTINCT, t_in(?categories),
                 t_out(?ancestors), t_max(4)) }
} ORDER BY DESC(?classes)
```

   For every DBpedia resource, `%URI%` is replaced with the DBpedia resource URI. The query gathers YAGO types up to the root of the hierarchy, and categories up until 4 ancestors. This query uses inferencing functions that are specific to the Virtuoso[2] triple store implementation, so a local query based on Jena stores will look differently. We currently do local queries without inferencing but with a large union of selects. The full collection of classes/categories obtained is persistently stored as an RDF *domain model*. In this model, each unique class/category is an RDF resource with the occurrence count stored as property `rdf:value`.

[2]urlhttp://virtuoso.openlinksw.com/

5. The bootstrap model is subsequently used as input for the various pruning and optimization methods. The `DomainManager` class contains methods to manipulate the domain, such as low-occurrence and high-generality domain pruning. `DomainManager` extracts each of the four class/category types into separate hashmaps, with the class/category as key and occurrence number as value, so that each can be pruned separately.

   Each of our described approaches is implemented as a separate method of the controller. The pruning methods in the `DomainManager` are called as needed with the right parameters, and will return the pruned domains as sets of classes and categories. The different optimization methods then apply whatever else processing is needed on the domain sets, and create `Matcher` instances to test performance of a domain filter on the entities of the bootstrap model using some Spotlight matcher (in our case always a `Disambiguator`) to evaluate performance according to the bootstrap linkset description.

   The genetic algorithm was implemented with the JGAP[3] library, using the default configuration and the `Matcher` as fitness function (with f_1 as fitness). The chromosomes are represented by domain filters converted to binary arrays (each class is either in or out).

For every subsequent run through the tool, the bootstrap model and domain model can be loaded directly, and the first four steps can be skipped, saving a significant amount of time. The main bottleneck is then the generation of alignment instances – for a bootstrap linkset with size 1000, this translates to 1000 Spotlight disambiguations per instance. Since we are not concerned with speed in this work, we will not give exact benchmarks as to the speed-wise performance of the approach. However, to give a rough idea, for a run that is fully cached, it takes about 2 seconds to disambiguate and filter 1000 entities and calculate the resulting scores on modern hardware. If all surface forms still need to be disambiguated (locally), it takes about 10 seconds. If we still need to (locally) query classes/categories for all DBpedia resources, it can take up to around 3 minutes. If any of these tasks need to be performed with online datasets, then the network latency will be the determining factor for the time it takes to complete.

3.5 Conclusion

In this chapter we described an unsupervised, domain-aware approach to the matching of instances to DBpedia. First, the input dataset is bootstrapped to obtain a sample of high-confidence matches to DBpedia resources – a bootstrap linkset – from which the associated classes and categories are gathered. This domain selection may be carefully pruned to discard uninformative or overly broad classes/categories. It was shown that, given the right bootstrap linkset format, there exists a strong correlation between performance of a matcher according to the bootstrap linkset and an actual reference alignment. Furthermore, this strong correlation is consistent across vastly different datasets, thereby proving its robustness. At the end of the chapter, some optimization

[3]`http://jgap.sourceforge.net/`

approaches to exploit this fact in order to improve the quality of the domain filter were introduced.

Now that we have formulated the full approach and proved the validity of the assumptions on which it is built, what remains is the evaluation of the performance of the optimization methods proposed. This is done in the following chapter and will be based on the use cases introduced in section 2.4 and illustrated throughout this chapter.

Chapter 4

Evaluation

In this chapter we evaluate the different methods that were devised in the approach chapter. Evaluation of ontology matching approaches is typically done by comparison of attained precision and recall scores with those of some baseline approach, or other matching systems that perform the same task. Since there is no other ontology matching system readily available that specializes in the task of creating links to DBpedia, the evaluation will primarily be done with regard to a baseline approach. We compare the performance of each of our approaches with a baseline approach: matching using DBpedia Spotlight without a domain filter. The approaches are also compared among each other. We do this for different source datasets, i.e. the use cases that were defined in section 2.4: Last.fm artists and UMLS terms. We calculate the precision, recall and F_1-scores for each approach. Insight is given into how the final scores are achieved by visualizing the approaches with graphs and tables.

This chapter is organized as follows. First we describe the experimental setup by which the results were achieved in more detail. Next, in 4.2, we list and briefly explain each evaluation strategy that we are going to compare, and display and discuss their individual results. The results are summarized and discussed in section 4.3. The section that follows shows results on some miscellaneous tests and refinements on the approach, such as good defaults for the matching phase and how different initial domain collection sizes will affect the result. Lastly, in section 4.5, we demonstrate the semantic enrichment of medical simulators, an example application that may benefit from the devised approach.

4.1 Experimental setup

In the approach chapter (3), we showed that the highest correlation between matching scores of an alignment instance $A_{R'}$ according to the bootstrap linkset R' and matching scores of an alignment instance A_R according to the actual reference alignment R is obtained if we consider both positive and negative matches in a 500/500 ratio (i.e. $R'_{500/500}$), context for each entity, and register only the first possible link in the match candidate ranking for each entity. For all evaluations henceforth, this will be the strategy used.

As mentioned in section 2.4, all experiments are performed two-fold – once on the EventMedia Last.fm dataset, and once on the UMLS. For Last.fm, we take all Artist resources – that is, all subjects `?artist` for which `?artist rdf:type foaf:Agent`. For UMLS, we take all subjects `?term` for which `?term rdf:type umls-semnetwork:{class}`, where *class* is the corresponding class for which we evaluate instances. We initially evaluate performance for two types of UMLS data: entities contained in class “Pathologic Function” (or any of its sub-classes), and entities contained in its sibling class “Physiologic Function” (refer to appendix B for a full view of the UMLS class hierarchy).

All optimization approaches derived in chapter 3 will be tested for all three datasets. Additionally, we determine two baselines that do not use domain information so that we can show whether our method can provide an improvement over conventional approaches. Performance for each approach is evaluated by comparison with the manually created reference alignments to obtain a precision score *Precision*, a recall score *Recall* and an F-score F_1. Therefore, for the actual matching step after a domain filter has been derived, we can limit ourselves to matching just those entities that are in these reference alignments (1000 for Last.fm, 500 for each UMLS class), i.e. for the evaluation we generate alignment instances A_R where R is the corresponding reference alignment. For comparison purposes, only an exact label matching phase is performed, with Spotlight parameters confidence (measure of similarity) $c = 0.0$, support (measure of popularity) $s = 0$ and the best-k, $k = 2$ candidates.

As an additional test, we check how performance on UMLS is affected if we try to match with a domain filter derived from class “Phenomenon or Process,” a common super-class of the UMLS classes Pathologic Function and Physiologic Function. By doing this we can show concretely whether there is value in splitting up the task of aligning larger ontologies such as UMLS by matching each class separately. For this class, we take a bootstrap linkset $R'_{1000/1000}$, twice as large as the two sub-classes in order to compensate for the increase in broadness. We also collect the classes and categories of 300 DBpedia resources of this bootstrap linkset instead of the usual 100. For evaluation, we record three separate scores: (1) the F_1-score according to the reference alignment of the Pathologic Function class, (2) the F_1-score according to the reference alignment of the Physiologic Function class, and (3) a combination of both, where we simply take the union of the reference alignments of both subclasses to get a reference alignment for Phenomenon or Process, with 1000 links.

Some additional refinements to the approach that we will explore are what we can choose as a good default for the best-k parameter; how the application of a second, partial match phase will influence the results, and what the best Spotlight confidence and support values c_2 and s_2 for this phase should be; and the effect of different class/category collection numbers on the final results. Lastly, the approach is tested on ImREAL medical simulators. This is a slightly different case from ontology matching, as the source data bears more resemblance to free text than the concretely labeled entities in ontologies. The approach is adapted slightly to cope with this, and compared with domain-unaware baselines.

4.2 Evaluation strategy and results

In this section we list all approaches we take, the exact parameters we use and the results by running them on the use cases. We divide the approaches into four categories:

baselines, basic strategies, metaheuristics and problem-specific heuristics. Each category of approaches is illustrated in a separate section. All approaches are applied to the Last.fm Artist dataset and the two UMLS datasets. A list of all categories and approaches, with section numbers where they are discussed, is as follows:

- *Baselines* (section 4.2.1)
  - True baseline: no domain filter
  - Optimal baseline: no domain filter, optimal matching settings
- *Basic strategies* (section 4.2.2)
  - Random selection
  - High-generality domain pruning
- *Metaheuristics* (section 4.2.3)
  - Genetic algorithm
- *Problem-specific heuristics* (section 4.2.4)
  - Broadness-based domain optimization
  - Precision-based domain optimization

4.2.1 Baselines

As a baseline, we try to match without considering a domain filter at all, i.e. the bootstrap and domain optimization steps are not applied, and the matching is performed as-is, relying only on similarity measures. Such a baseline is essential in showing the added value of a domain-aware matching approach. However, it may depend on the dataset which matching settings are optimal without a domain restriction, and we cannot find this out without performing an evaluation on some manual reference links. For the purpose of this evaluation, we will distinguish between a *true baseline*, and an *optimized baseline*.

True baseline For the true baseline, matching is kept consistent with the domain filter-based approaches. This means that we set Spotlight parameters confidence $c = 0.0$ and support $s = 0$. Which top-k candidates to consider is not relevant without a domain filter: the first candidate is picked no matter what, as we cannot tell whether it is within the domain of interest or not.

Optimized baseline For the second, optimized baseline, Spotlight's matching parameters are optimized based on F_1 by using the manually created reference alignments. Although this is not normally possible if we assume a fully automated approach, it will serve as a point of reference in the comparison with approaches that do employ domain filters, as it provides a theoretical upper-bound on performance for matching with DBpedia Spotlight without a domain filter.

We need to optimize confidence c and support s. This is done by hillclimbing towards the (near) optimal solution. We determine this by evaluating the resulting

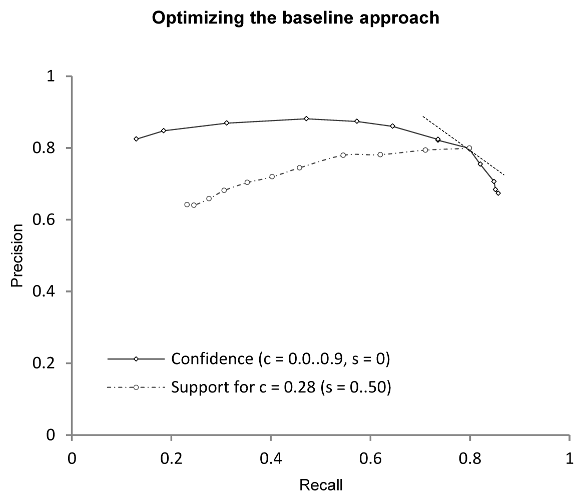


Figure 4.1: Hillclimbing to optimize the baseline F_1-score for the Last.fm use case. The point marked by the diagonal corresponds to the optimal F_1-score ($c = 0.28, s = 0$).

matches against the reference alignment R. First, we keep s fixed at 0 and k at 1 and vary c between 0 and 1 in steps of 0.1. Next, we take the value of c that provided the highest F_1, as well as a few below it: increasing either c or s is analogous to raising precision and lowering recall, but the ratio at which this happens is not always the same for both. At points this ratio may be more favorable for increasing the F-score with c than with s or vice versa. We vary s between 0 and 50 in steps of 5 for each of the c we chose to check. We take this relatively low range of inlinks due to the nature of both datasets, which largely consists of fairly obscure entities that are likely not often linked to. We settle on whichever combination of c and s gives the best F_1.

A graph based on the Last.fm Artist dataset showcasing the precision and recall for the different values of c with s at 0, and the different values of s with the optimal c, is depicted in figure 4.1. For this set we arrive at $c = 0.28$ and $s = 0$ for the optimal values. We apply this strategy analogously with the UMLS datasets, and arrive at optimal values of $c = 0.2$ and $s = 0$ for both. Table 4.1 shows the final results for both the true and optimized baselines.

4.2.2 Basic strategies

Two light-weight, basic strategies are discussed that may obtain a good result for relatively little computation time. We introduce a *random selection* strategy, and a strategy for applying *high-generality domain pruning* in a way that we can optimize the result.

Random selection A very basic optimization approach is to simply take the best result from a random sample. The low-occurrence domain pruning (LDP) strategy (section 3.3.4.1) is executed first to filter the selection of the least used classes/categories, with the default parameter settings of $r_d = 1.0$, $r_y = 1.0$, $t_c = 10$ and $t_f = 6$ to obtain a relatively wide initial domain selection. From this domain, we generate 100 sub-selections randomly. The randomness is achieved by iterating through the pruned selection and deciding for each class/category in the selection whether it should be

Table 4.1: Results for the baselines on the Last.fm and UMLS datasets. Confidence c and support s are included to show how we arrived at the optimized baselines.

Last.fm artists

Approach	c	s	*Precision*	*Recall*	F_1
True baseline	0	0	0.673	0.857	0.754
Optimized baseline	0.28	0	0.798	0.798	0.798

UMLS Pathologic Function

Approach	c	s	*Precision*	*Recall*	F_1
True baseline	0	0	0.950	0.941	0.946
Optimized baseline	0.2	0	0.960	0.941	0.950

UMLS Physiologic Function

Approach	c	s	*Precision*	*Recall*	F_1
True baseline	0	0	0.837	0.975	0.901
Optimized baseline	0.2	0	0.852	0.975	0.909

included or not with a probability of $\frac{1}{2}$. We then calculate f_1-score performance according to the bootstrap linkset R' for each and choose the best-performing selection as final domain filter.

High-generality domain pruning (HDP) We apply both the low-occurrence and high-generality domain pruning strategies (3.3.4) for varying parameter settings, and choose as domain filter the result that provides the highest f_1-score. Altering the parameters is needed in order to test as many different final domain selections as possible. For the LDP step, we define as possible parameter values $r_d = \{1.0, 2.0\}$, $r_y = \{1.0, 2.0\}$, $t_c = \{0, 5, 10\}$ and $t_f = \{0, 3, 6\}$, and for HDP $p = \{70, 80, 90\}$. The resulting pruned collection is tested against the bootstrap linkset for every $2 \times 2 \times 3 \times 3 \times 3 = 108$ possible parameter combinations.

Results for the basic strategies are shown in table 4.2, which shows the matching scores and corresponding domains (trimmed in case they are too large) for the best performing results. We see that the random selection strategy tends to produce large domains. This is no surprise, since after the LDP step we are still left with 40 classes in the Last.fm Artist case and 43 for both UMLS datasets. This leaves an expected size of

$$\mathrm{E}[X] = \sum_{i=1}^{n} \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1 = \frac{1}{2} n, \tag{4.1}$$

with X the classes as random variable that can take the value 1 (included) or 0 (excluded) with equal probability of $\frac{1}{2}$ and n the collection size. This gives 20 for Last.fm and 21.5 for UMLS.

Table 4.2: Best results for the basic strategies on the Last.fm and UMLS datasets.

Last.fm artists

Approach	*Precision*	*Recall*	F_1	Domain
Random	0.722	0.865	0.787	DBpedia: MusicalArtist, DBpedia: Person, YAGO: Musician, YAGO: Organization, YAGO: Whole, YAGO: Singer, YAGO: Person, and 11 more
HDP	0.935	0.879	0.906	DBpedia: MusicalArtist, DBpedia: Band, YAGO: Musician, YAGO: MusicalOrganization, YAGO: 2000sMusicGroups, YAGO: Singer, YAGO: LivingPeople, Category: Musicians_by_nationality, Category: Musicians_by_genre

UMLS Pathologic Function

Approach	*Precision*	*Recall*	F_1	Domain
Random	0.955	0.931	0.943	YAGO: Illness, YAGO: Syndrome, YAGO: Idea, YAGO: Condition, YAGO: Complex, YAGO: Abstraction, YAGO: Disease, and 14 more
HDP	0.994	0.877	0.932	DBpedia: Species, DBpedia: Disease, YAGO: Syndromes, YAGO: Disease, YAGO: Disorder, Freebase: /medicine, Freebase: /medicine/disease, Freebase: /medicine/icd_9_cm_classification

UMLS Physiologic Function

Approach	*Precision*	*Recall*	F_1	Domain
Random	0.902	0.928	0.915	YAGO: Illness, YAGO: Syndrome, YAGO: Idea, YAGO: Condition, YAGO: Complex, YAGO: Abstraction, YAGO: Disease, and 14 more
HDP	0.981	0.883	0.930	DBpedia: AnatomicalStructure, YAGO: Enzyme, YAGO: Protein, YAGO: Process, Category: Biology

The HDP strategy tends to select much smaller domains, and performs significantly better than a random selection for Last.fm. There is no significant difference in performance between the two for UMLS.

4.2.3 Metaheuristics

Metaheuristics have been explained in section 3.3.6.1: they are generally applicable optimization methods that work without any knowledge of the underlying problem. A problem does usually need to be converted into a format that the metaheuristic can handle. We apply a *genetic algorithm* for the optimization of a domain filter.

Genetic algorithm We adapt our problem to a format that can be processed by a genetic algorithm. First, low-occurrence domain pruning is applied with default settings to limit the search space to a processable size. For Last.fm, we end up with a domain D of size 40. D is then encoded as a chromosome, i.e. a bit array of 40 bits, where each bit represents whether the corresponding class/category is included or not. As

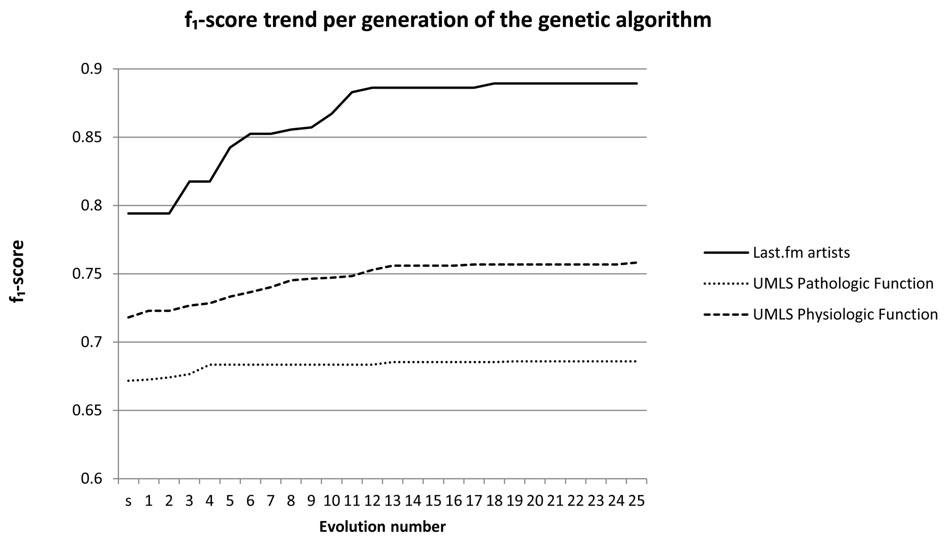


Figure 4.2: Evolution of the f_1-score for the genetic algorithm. Note: s is the best score of the (randomly generated) starting population.

fitness function to check the quality of a chromosome (domain filter) D, we generate a domain-restricted alignment instance $A_{R',D}$ and calculate f_1 with regard to R'.

Genetic algorithms are highly customizable: they have many different parameters, such as the initial population size, selection strategy and proportion of the population that is taken to the next generation, and the probabilities for crossover and mutation. Since discovering the best values for any specific problem is difficult and time-consuming, we decide to simply stick with a starting population of 100 chromosomes, and the default values for selection, crossover and mutation that came with the out-of-the-box genetic algorithm implementation employed for this experiment[1]. There may be other setups that can more quickly arrive at an optimal result, but we do not consider performance to be crucial for these experiments.

Given the default implementation, for each generation, the fittest 90% of the population is taken to the next generation, and for each chromosome in every evolution, there is a 35% chance for crossover with another chromosome (one or more included/excluded classes/categories are switched to obtain two new chromosomes), and a 1/12 chance for mutation (a previously excluded class/category in a D is now included, or vice versa). The algorithm is run for 50 generations, with the fittest remaining solution chosen as the result.

Figure 4.2 shows the evolution of the f_1-score of the fittest chromosome per generation for all three datasets until the 25th generation, and table 4.3 shows the matching scores and corresponding domains for milestone generations up until the 25th generation. Everything past this 25th generation is omitted as the algorithm had already reached its optimum, and there was no longer any change in f_1-score. Note that whenever there was more than one unique domain filter for the same f_1-score, we prioritized the largest domain filter.

We see that for the Last.fm Artist dataset, the evolution shows the most variabil-

[1] `http://jgap.sourceforge.net/`

ity, rising from 0.774 to 0.889 over the course of 25 generations. The Physiologic Function dataset rises more slowly and the Pathologic Function dataset does not show much change at all. This implies that the Last.fm dataset has much more potential for optimization than the other two, with large differences between inefficient domain filter selections and better selections. The tables show a steady increase of the F_1-score along with the f_1-score for Last.fm, although it decreases for the last generation. This can be accredited to an imperfect correlation between f_1 and F_1: the more minor the increase in f_1, the higher the chance that F_1 does not rise along with it. It can be seen that the correlation is steady for more substantial increases (i.e. from generation s to 15). This effect was illustrated clearly in figure 3.7 of section 3.3.5.2.

For the UMLS Pathologic Function dataset, values are as expected from looking at the evolution graph: a minor gradual increase in f_1 alongside a minor gradual increase in F_1. The Physiologic Function values are less satisfying: while the f_1 increase can be considered significant, corresponding F_1 are not evolving as desired. This is again likely due to an imperfect correlation.

Table 4.3: Results for several milestone generations of the genetic algorithm on the Last.fm dataset. Note: s is the best result of the starting population.

Last.fm artists

Gen.	f_1	*Precision*	*Recall*	F_1	Domain
s	0.794	0.791	0.873	0.830	DBpedia: Band, DBpedia: Person, YAGO: Group, YAGO: SocialGroup, YAGO: LivingPeople, YAGO: Singer, YAGO: Person, and 6 more
5	0.842	0.919	0.876	0.897	DBpedia: Band, DBpedia: Person, YAGO: Entertainer, YAGO: 2000sMusicGroups, YAGO: LivingPeople, YAGO: Singer, Category: Musicians_by_genre, Category: Musicians
10	0.867	0.936	0.882	0.908	DBpedia: Band, YAGO: Entertainer, YAGO: 2000sMusicGroups, YAGO: LivingPeople, YAGO: Singer, Category: Musicians_by_genre, Category: Musicians, Freebase: /music/artist
15	0.886	0.961	0.879	0.918	DBpedia: Band, DBpedia: Artist, YAGO: Singer, Category: Musicians_by_genre, Category: Musicians, Freebase: /music/musical_group, Freebase: /music/artist
20-50	0.889	0.960	0.868	0.912	DBpedia: Band, DBpedia: Artist, YAGO: MusicalOrganization, YAGO: Singer, Category: Musicians_by_genre, Freebase: /music/musical_group, Freebase: /music/artist

UMLS Pathologic Function

Gen.	f_1	*Precision*	*Recall*	F_1	Domain
s	0.673	0.955	0.931	0.943	DBpedia: AnatomicalStructure, DBpedia: Nerve, YAGO: Illness, YAGO: Syndromes, YAGO: Whole, YAGO: Disorder, YAGO: Complex, and 14 more
5	0.677	0.964	0.931	0.947	DBpedia: AnatomicalStructure, YAGO: Illness, YAGO: Syndromes, YAGO: Whole, YAGO: Condition, YAGO: Disorder, Category:Health_sciences, and 5 more

Table 4.3: Continued from the previous page. Results for several milestone generations of the genetic algorithm on the Last.fm dataset. Note: s is the best result of the starting population.

10	0.678	0.964	0.931	0.947	DBpedia: AnatomicalStructure, DBpedia: Nerve, YAGO: Illness, YAGO: Syndromes, YAGO: Whole, YAGO: Condition, YAGO: Disorder, and 5 more
15	0.679	0.964	0.931	0947	DBpedia: AnatomicalStructure, DBpedia: Nerve, YAGO: Syndromes, YAGO: Whole, YAGO: Disorder, Category:Health_sciences, Category: Biology, and 3 more
20	0.680	0.964	0.931	0.947	DBpedia: AnatomicalStructure, DBpedia: Nerve, YAGO: Syndromes, YAGO: Whole, YAGO: Disorder, Category: Biology, Freebase: /medicine/risk_factor, Freebase: /medicine, Freebase: /medicine/disease_cause
25-50	0.686	0.974	0.931	0.952	DBpedia: AnatomicalStructure, DBpedia: Nerve, YAGO: Syndromes, YAGO: Complex, YAGO: Disorder, Category: Biology, Freebase: /medicine/risk_factor, Freebase: /medicine, Freebase: /medicine/disease_cause, Freebase: /medicine/disease, Freebase: /medicine/icd_9_cm_classification

UMLS Physiologic Function

Gen.	f_1	*Precision*	*Recall*	F_1	Domain
s	0.718	0.891	0.953	0.921	DBpedia: AnatomicalStructure, YAGO: Compound, YAGO: Chemical, YAGO: Unit, YAGO: Relation, YAGO: Matter, YAGO: Part, and 15 more
5	0.733	0.952	0.930	0.941	YAGO: Compound, YAGO: Chemical, YAGO: Process, YAGO: Thing, YAGO: Substance, YAGO: Enzyme, Protein, and 9 more
10	0.747	0.964	0.908	0.935	YAGO: Compound, YAGO: Chemical, YAGO: Unit, YAGO: Activator, YAGO: Process, YAGO: Thing, YAGO: Substance, and 6 more
15	0.756	0.979	0.891	0.933	YAGO: Compound, YAGO: Chemical, YAGO: Process, YAGO: Thing, YAGO: Substance, YAGO: Macromolecule, YAGO: Enzyme, and 3 more
20	0.757	0.979	0.891	0.933	YAGO: Compound, YAGO: Chemical, YAGO: Process, YAGO: Thing, YAGO: Substance, YAGO: Protein, YAGO: Enzyme, and 2 more
25-50	0.758	0.979	0.891	0.933	DBpedia: AnatomicalStructure, YAGO: Unit, YAGO: Protein, YAGO: Process, YAGO: Macromolecule, YAGO: Enzyme, YAGO: Thing, YAGO: Substance, Category: Biology, Freebase: /media_common/quotation_subject

4.2.4 Problem-specific heuristics

Metaheuristics optimize problems in a generic way, and can therefore be applied to most optimization problems. The penalty for this generality is increased computation time complexity of the problem solver. Another way to tackle an optimization problem is to look deeper into the nature of the problem and exploit features specific to it in order to obtain better optimal results, or to arrive at an optimum faster. In section 3.3.6.2, we had introduced two problem-specific heuristics, *broadness-based domain optimization* and *precision-based domain optimization*, that we can apply to the derivation of a domain filter. In this section we apply them to the datasets.

Broadness-based domain optimization (BBO) The broadness-based domain optimization approach explained in section 3.3.6.2 is applied as an additional step to the HDP-based basic strategy described in section 4.2.2. As with the HDP approach, there are 108 total domain combinations. With the BBO algorithm, for each of these domain combinations D_i, there are $|D_i| + 1$ alignment instances $A_{R',D}$ generated (see Algorithm 1 in section 3.3.6.2). The domain filter D_{best} that provided the highest recorded f_1-score is taken as the result (if more than one combinations share the maximum f_1-score, the result with the largest domain filter is chosen).

Precision-based domain optimization (PBO) Precision-based domain optimization is applied just as described in section 3.3.6.2. We use the default LDP parameters to obtain the initial domain D, then for each resulting class/category, we generate alignment instances A_{R',D_1} where each D_1 is a single-class domain filter. For each A_{R',D_1}, $Precision'$ is calculated and the classes in D are sorted according to this $Precision'$ score. Subsequently, for $k = 1$ to $|D|$, we generate an A_{R',D_k} where D_k is a domain filter containing the top-k classes from the sorted class list. Table 4.4 shows part of this list (the top ten) for all three datasets. For every k, f_1 is evaluated, and the D_k providing the highest f_1 is chosen as final result. If more than one domain filter lead to the same highest f_1, the largest of them is chosen.

Table 4.5 shows the precision, recall and F_1 scores of the best results, along with the associated domains, of the problem-specific heuristic-based strategies. For the broadness-based optimization approach, we appear to always end up with very compact domain filters. This is because the domain is pruned rather aggressively with this approach: after applying low-occurrence and high-generality pruning, we attempt to prune what remains even further by eliminating classes that are even slightly too broad. This is reflected in the precision scores, which reach as high as 1, for the Pathologic Function class.

For the precision-based optimization approach, the Artist dataset gave the best result for a domain filter containing the top 9 classes/categories, for Pathologic Function the top 26, and for Physiologic Function the top 10. Which type of classes get used depends heavily on the domain of the underlying dataset: in table 4.4, we see that the top ten for Last.fm is a diverse mix, and the most precise classes for Physiologic Function are categories. In other words, artists can be described well with all schema types, while pathologic terms are biased towards YAGO and physiologic terms biased towards categories. This difference could be related to the manner in which each respective schema has been generated, but the answer to this question is outside the scope of this thesis.

We see that in all cases, precision-based outperforms broadness-based in terms of F_1 – apparently, BBO is too heavily biased towards precision, to the point that recall suffers too much. This results in an unbalanced result and thus lower F_1-scores.

Table 4.4: Top ten classes sorted by virtual precision values *Precision'* when using these classes as single-class domain filters D_1 for the generation of A_{R',D_1}.

Last.fm artist

Class/category	*Precision'*	*Recall'*	f_1
DBpedia:Band	0.831	0.582	0.685
Freebase:/music/musical_group	0.828	0.626	0.713
Freebase:/music/artist	0.818	0.952	0.880
YAGO:MusicalOrganization	0.807	0.260	0.393
YAGO:2000sMusicGroups	0.798	0.142	0.241
Category:Musicians	0.776	0.742	0.759
Category:Musicians_by_nationality	0.776	0.768	0.772
Category:Musicians_by_genre	0.775	0.642	0.702
Freebase:/music/group_member	0.741	0.280	0.406
Freebase:/music	0.738	0.960	0.835

UMLS Pathologic Function

Class/category	*Precision'*	*Recall'*	f_1
DBpedia:Nerve	1.000	0.012	0.024
YAGO:Complex	0.807	0.092	0.165
YAGO:Syndrome	0.807	0.092	0.165
YAGO:Syndromes	0.796	0.086	0.155
YAGO:Whole	0.793	0.092	0.165
YAGO:Disorder	0.636	0.196	0.300
YAGO:IllHealth	0.599	0.254	0.357
YAGO:PathologicalState	0.254	0.768	0.357
YAGO:Disease	0.592	0.232	0.333
YAGO:Illness	0.589	0.232	0.333

UMLS Physiologic Function

Class/category	*Precision'*	*Recall'*	f_1
Category:Cell_biology	0.725	0.521	0.606
Category:Biochemistry	0.673	0.513	0.582
Category:Biology	0.631	0.513	0.582
Category:Chemistry	0.612	0.547	0.578
YAGO:Process	0.611	0.234	0.451
Category:Natural_sciences	0.590	0.679	0.631
Freebase:/media_common/quotation_subject	0.552	0.034	0.064
YAGO:Enzyme	0.543	0.404	0.075
YAGO:Catalyst	0.543	0.404	0.075
YAGO:Activator	0.543	0.404	0.075

Table 4.5: Best results for the problem-specific heuristic-based domain optimization approaches on the Last.fm and UMLS datasets. Note: BBO = broadness-based optimization, PBO = precision-based optimization.

Last.fm artist

Approach	*Precision*	*Recall*	F_1	Domain
BBO	0.965	0.837	0.897	DBpedia: MusicalArtist, DBpedia: Band, Freebase: /music/group_member/, Freebase: /music/musical_group/
PBO	0.964	0.884	0.922	DBpedia: Band, YAGO: MusicalOrganization, YAGO: 2000sMusicGroups, Category: Musicians, Category: Musicians_by_nationality, Category: Musicians_by_genre, Freebase: /music/musical_group, Freebase: /music/group_member, Freebase: music/artist

UMLS Pathologic Function

Approach	*Precision*	*Recall*	F_1	Domain
BBO	1.000	0.852	0.920	DBpedia: Species, DBpedia: Disease, DBpedia: Nerve, YAGO: IllHealth, YAGO: Disease, YAGO: Disorder, Freebase:/medicine/disease, and 4 more
PBO	0.964	0.936	0.950	DBpedia: Nerve, DBpedia: Disease, YAGO: Complex, YAGO: Syndrome, YAGO: Syndromes, YAGO: Whole, YAGO: Disorder, and 19 more

UMLS Physiologic Function

Approach	*Precision*	*Recall*	F_1	Domain
BBO	0.979	0.891	0.933	DBpedia: AnatomicalStructure, YAGO: Enzyme, YAGO: Protein, YAGO: Process, Category: Biology, Freebase: /media_common/quotation_subject
PBO	0.965	0.925	0.945	YAGO: Enzyme, YAGO: Process, YAGO: Activator, YAGO: Catalyst, Category: Biology, Category: Cell_biology, Category: Natural_sciences, Category: Biochemistry, Category: Chemistry, Freebase: /media_common/quotation_subject

4.3 Summarized experimental results

The main results for both use cases are summarized and listed in this section. f_1 results are included to be able to more accurately compare raw f_1-based performance between approaches – as explained before, while the correlation between f_1 and F_1 has been shown to be very high, it is not a *perfect* correlation, and hence it can theoretically occur that for two pairs of $\langle f_{1,x}, F_{1,x} \rangle$, $f_{1,i} \geq f_{1,j} \not\Rightarrow F_{1,i} \geq F_{1,j}$. It may also be dependent on the optimization method used to derive the f_1. Aside from the matching scores, we also list computation complexity in terms of the number of alignment instances $A_{R'}$ we need to generate on average for each approach, so that we also have a performance measure that we can bring into the comparison. Each approach also includes one A_R generation, i.e. a matching pass on the entities of the actual reference alignment. Note that in an actual usage scenario, this would be replaced with a matching pass on the entities of the entire source dataset.

Table 4.6: Summarized results for each approach on all datasets, sorted by F_1-score. Note: n_{LDP} and n_{HDP} refer to the size of the domain filter after applying the LDP resp. HDP pruning strategies. n is variable depending on the chosen parameters, but is typically around 30 to 40 for n_{LDP} and 5 to 15 for n_{HDP}. The multiplier for the genetic algorithm is based on the crossover/mutation rate per generation, for 50 generations.

Last.fm artist

Approach	f_1	*Precision*	*Recall*	F_1	Complexity
Precision-based opt.	0.884	0.964	0.889	0.925	$2n_{LDP} \cdot A_{R'} + A_R$
Genetic algorithm	0.889	0.960	0.868	0.912	$8600 \cdot A_{R'} + A_R$
HDP	0.864	0.935	0.879	0.906	$108 \cdot A_{R'} + A_R$
Broadness-based opt.	0.890	0.965	0.837	0.897	$108n_{HDP} \cdot A_{R'} + A_R$
Optimized baseline	n/a	0.798	0.798	0.798	A_R
Random selection	0.768	0.722	0.865	0.787	$100 \cdot A_{R'} + A_R$
True baseline	n/a	0.673	0.857	0.754	A_R

UMLS Pathologic Function

Approach	f_1	*Precision*	*Recall*	F_1	Complexity
Genetic algorithm	0.686	0.974	0.931	0.952	$8600 \cdot A_{R'} + A_R$
Precision-based opt.	0.674	0.964	0.936	0.950	$2n_{LDP} \cdot A_{R'} + A_R$
Optimized baseline	n/a	0.960	0.941	0.950	A_R
True baseline	n/a	0.950	0.941	0.946	A_R
Random selection	0.670	0.955	0.931	0.943	$100 \cdot A_{R'} + A_R$
HDP	0.672	0.994	0.877	0.932	$108 \cdot A_{R'} + A_R$
Broadness-based opt.	0.679	1.000	0.852	0.920	$108n_{HDP} \cdot A_{R'} + A_R$

UMLS Physiologic Function

Approach	f_1	*Precision*	*Recall*	F_1	Complexity
Precision-based opt.	0.741	0.965	0.925	0.945	$2n_{LDP} \cdot A_{R'} + A_R$
Genetic algorithm	0.758	0.979	0.891	0.933	$8600 \cdot A_{R'} + A_R$
Broadness-based opt.	0.758	0.979	0.891	0.933	$108n_{HDP} \cdot A_{R'} + A_R$
HDP	0.753	0.981	0.883	0.930	$108 \cdot A_{R'} + A_R$
Random selection	0.719	0.902	0.928	0.915	$100 \cdot A_{R'} + A_R$
Optimized baseline	n/a	0.852	0.975	0.909	A_R
True baseline	n/a	0.837	0.975	0.901	A_R

The resulting virtual F-score f_1 and actual precision *Precision*, recall *Recall* and F-scores F_1 for each approach are displayed in table 4.6. For Last.fm, a clear improvement can be seen over the baseline for any of the domain filter-based approaches barring a random selection. The best performing approach is the precision-based optimization, giving an F_1-score of 0.922, which is $0.922 - 0.798 = 0.124$ (12.4%) higher than the optimized baseline and $0.925 - 0.754 = 0.171$ (17.1%) higher than the true baseline in terms of absolute F_1-score. Differences amongst the optimization approaches are relatively small – the most significant difference exists between the precision-based method and the broadness-based method ($0.925 - 0.897 = 0.025$ (2.8%)). Additionally, it shows that a higher f_1 does not always lead to a higher F_1 when differences between f_1 are small, since the correlation is not perfect. The correlation *does* clearly hold for larger differences, as seen when comparing the random selection with any of the top four approaches.

The summarized results for the UMLS datasets “Pathologic Function” and “Physiologic Function” are displayed in the tables below the Last.fm Artist results, also in 4.6. For Pathologic Function, there is no clear difference to be seen between any of the approaches. This can likely be attributed to the fact that the entities in this class are highly specific terms (a significant portion consists of medical terminology for diseases), that are very unambiguous, so that correct matches to DBpedia resources are easily found. Here, we see that a domain filter can potentially even have an adverse effect and lead to worse performance than the baseline. This is because entities that do not have very specific type information in DBpedia would still yield a correct match with high probability due to the unambiguity, but may get excluded from a filter that is too strict.

For Physiologic Function, there is a clear improvement over the baselines again, although it is less significant than in the Last.fm case. This dataset contains more ambiguity than the Pathologic Function dataset (e.g. common mental processes such as “Recognition,” and genes with abbreviations that are commonly used to refer to other things), but is still quite specialized, therefore the impact of a domain restriction is not as obvious as for very ambiguous entity labels such as artist and band names.

4.3.1 Margin of error

In [17], it is shown that given a sample of the full set of matches (i.e. our reference alignment R), and p the true proportion of samples produced that is correct (which is unknown), n the size of the sample used to approximate p (the size of the reference alignment, in our case 1000 for the Last.fm Artist set and 500 for each UMLS set) and $\hat{P}$ the approximation of p based on the reference alignment, then this approximation lies in the interval

$$\hat{P} \in [p - \delta, p + \delta] \text{ where } \delta = \frac{1}{\sqrt{n}} \tag{4.2}$$

with 95% confidence. δ is thus the 95% confidence margin of error for the result. For example, if one result falls within this range of another result, then these two results do not differ sufficiently from each other to be able to state with certainty that one is better than the other. We can apply this calculation on the resulting F_1-scores for each approach to verify their statistical significance: we want to make sure that a good result according to R leads to a good result on the full dataset, and that an improvement of an approach over other approaches and the baselines is actually significant enough to draw conclusions about them.

For the Last.fm dataset, we have $n = 1000$, so the margin of error δ becomes $\frac{1}{\sqrt{1000}} = 0.032$. We can therefore conclude that all optimization approaches aside from the random selection yield a significant improvement over the baselines – the smallest difference is between the optimized baseline and the broadness-based approach, which is $0.897 - 0.798 = 0.099$. Differences among optimization approaches are at most 0.028 (between precision-based and broadness-based), so we cannot state that one optimization approach is better than the other with 95% confidence.

For the UMLS classes, we are dealing with reference alignments of size 500 for both, so here the 95% confidence margin of error for both is $\frac{1}{\sqrt{500}} = 0.045$. For Pathologic Function, none of the optimization approaches are significantly better than the baselines. For Physiologic Function, the largest difference is between the true baseline and precision-based optimization, which is a difference of 0.044. This would be borderline insignificant if we considered this as an individual occurrence, but we can see that *each* optimization approach in fact outperforms the baseline by at least 0.029. Since the optimization approaches consistently perform better, this suggests that the improvements over the baseline are significant.

4.4 Approach refinements

In this section we present some miscellaneous refinements to the approach. First, we look at how performance when deriving a domain from a super-class compares to performance on the sub-classes of this class. Since we already obtained results for UMLS classes "Pathologic Function" and "Physiologic Function," we take their mutual super-class "Phenomenon or Process." Secondly, an appropriate value for k is discussed. Recall that k refers to the best-k candidates to consider in the matching phase. Up until now we have kept this value fixed at $k = 2$ for evaluation purposes, but it is not clear whether this is also the optimal value. We then show how we can potentially improve upon the matching by introducing a second, partial matching phase. We also investigate the effect of a different volume of DBpedia resources from which to collect classes and categories, which had been kept constant at 100 resources until now. We show whether increasing this number will positively affect the results, and how small we can make this set without losing reliability for domain derivation.

4.4.1 Deriving the domain with a super-class

We apply the methods from the evaluation for the UMLS classes analogously to a super-class that contains both of these classes, Phenomenon or Process, to see how this affects the results. We use a larger bootstrap linkset and collect more classes and categories than usual to compensate for the increase in size of the dataset (198,665 entities, against 70,046 resp. 54,781 for the Pathologic and Physiologic Function subsets, see appendix B for details). This allows there to be enough bootstrap links from each of the various sub-classes. Once the domain filters have been derived, there are a number of evaluation choices: (1) we can use the reference alignment of the Pathologic

Table 4.7: Summarized results for a domain derived on the UMLS class Phenomenon or Process, with F_1-scores on the reference alignments for Pathologic Function (R_{path}), Physiologic Function (R_{phys}) and the union of both ($R_{path} \cup R_{phys}$). Results have been sorted by F_1 on the union. Values between brackets are the previously determined scores derived with their own bootstrap linksets.

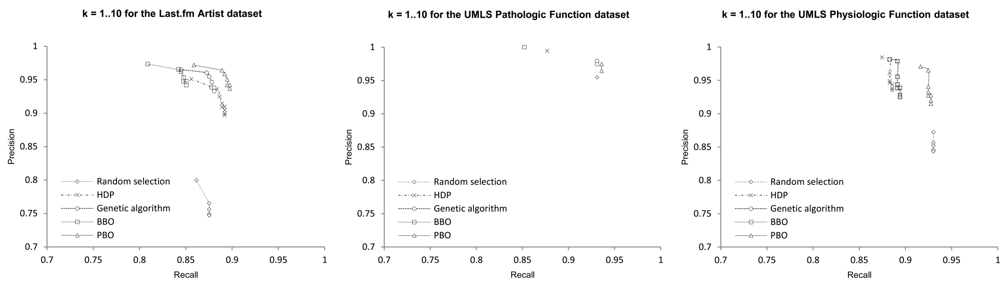
Approach	f_1	F_1 on R_{path}	F_1 on R_{phys}	F_1 on $R_{path} \cup R_{phys}$
Precision-based opt.	0.710	0.957 (0.950)	0.934 (0.945)	0.942
Genetic algorithm	0.712	0.957 (0.952)	0.931 (0.933)	0.941
HDP	0.699	0.955 (0.932)	0.931 (0.930)	0.940
Broadness-based opt.	0.699	0.955 (0.920)	0.931 (0.933)	0.940
Random selection	0.693	0.948 (0.943)	0.912 (0.915)	0.924
Optimized baseline	n/a	0.950	0.909	0.923
True baseline	n/a	0.946	0.901	0.916

Function class, which we will call R_{path}, to see how deriving a domain from a more general point of reference will influence the result of a single specific sub-domain; (2) we can do the same for Physiologic Function class, using R_{phys}; and (3) we can evaluate the union of both, $R_{path} \cup R_{phys}$, for the actual averaged quality over both datasets, i.e. the scores they would get if we were to match the full Phenomenon or Process class.

The results are displayed in table 4.7. The f_1 column shows the obtained scores using a bootstrap linkset $R'_{1000/1000}$ and collecting classes/categories for 300 of the positive matches. The second and third columns show the F_1 results on the two subdomains separately. The values within brackets are the scores that we found earlier, i.e. when using a bootstrap linkset that is derived from these sub-domains separately. Note that the baselines here are simply the values obtained earlier, since no domain filter is used here. The final column shows the combined result.

The most interesting result that this table shows is that a domain filter derived from a more general class consistently results in better matches for the Pathologic Function class, which we showed did not yield a significant improvement over the baseline when a domain filter is derived from this class itself due to its unambiguous nature. As explained before, this latter domain filter is too strict, potentially excluding related entities that do not contain specific enough domain information in DBpedia. A domain derived from a super-class is broader, therefore more of these types of entities may be included.

For the Physiologic Function class, scores are slightly worse than before, which is the expected result when having a broader domain filter for a domain that is (slightly) ambiguous, since we end up with more unrelated matches. When the union of both classes is considered, results are averaged as expected. Comparing these results to the previous F_1-scores (displayed in parentheses), we see that applying the HDP and broadness-based optimization approaches to the union of the two sub-domains gives a better overall result than applying them to the sub-domains separately. This is because these approaches apply very aggressive pruning, thereby excluding too many related



(a) k evolution for Last.fm Artist. (b) k evolution for UMLS Pathologic Function. (c) k evolution for UMLS Physiologic Function.

Figure 4.3: Precision and recall values for each approach for different values of k.

entities. It is the other way around for the precision-based optimization and genetic algorithm approaches.

4.4.2 Selecting the optimal best-k value

We can attempt to find out which setting for k would be most suitable. Recall that k decides the number of candidates to consider in the final matching stage *only*. It should therefore be noted that it is fully independent of the methods used to derive and execute the optimization approaches. So while it was kept at $k = 2$ for the duration of the procedure, this has not biased the approaches to work best with this value.

Although it is not possible to test and optimize this value in the same way as we did with the domain filter, it is possible to derive a robust default value by testing the match approaches for different values of k. We generate ten alignment instances A_R with each of the optimal domain filters derived (i.e. all approaches aside from the baselines), for each of the three main datasets, varying k from 1 to 10. The value of k that gives the #1 result the most often will be our default.

The results for each dataset have been plotted in figure 4.3. The graphs clearly show the tradeoff between precision and recall: as k grows, the precision goes down and recall rises. The absolute differences in F_1-scores are also quite minor – there is at most a 2-3% difference between the lowest and highest values. For the Last.fm Artist dataset, $k = 2$ gives the best F_1-score for all approaches except BBO, where $k = 3$ is very slightly better (by 0.03%). For the Pathologic Function dataset, the results remain the same for each approach and each k, except for PBO and the genetic algorithm, where there is a different result between $k = 1$ and $k = 2$. In both cases, $k = 1$ gives a slightly better result. For the Physiologic Function dataset, $k = 2$ performs best for each approach.

In the overwhelming majority of cases, $k = 2$ provides the highest F_1-score. We can therefore presume this to be a good default to select for other datasets as well.

4.4.3 Partial match phase

Once we have applied one of the approaches described in this and the previous chapter and created some matches to DBpedia, we can go over all source entities again, this time trying to match sub-entities that may be hidden within these entities. The rationale for this was described in section 3.3.7. DBpedia Spotlight offers an "annotate" feature that can do this for us. It works by first checking whether the full label given has an exactly matching surface form in its candidate index (this is essentially the "disambiguate" function). It then takes off the most left word of the label and the most right word of the label to end up with two *$(n-1)$-grams*, where n is the number of words in the full label. It then tries to disambiguate again for both. This process is repeated until the n-grams can no longer be split up any further. Note that it is possible to have more than one match for one label. Whether to actually consider candidates that are found matches or not is again done by thresholding on confidence and support.

Whether to apply a partial match phase or not depends on whether partial matches are actually wanted. For UMLS, for example, a term might refer to a specific type of disease, e.g. "Stage I Wilms' tumor." This exact term is not in DBpedia, but just "Wilms' tumor" is. Whether or not this is then still a proper match is open to interpretation. An alternative option is to consider it a related match rather than an exact match, and represent this accordingly (e.g. by creating links with the SKOS property `skos:closeMatch` rather than `skos:exactMatch`). In order to evaluate the partial match phase, we need to assume that partial matches are considered equivalent to full matches. The Last.fm Artist reference alignment was created with this assumption in mind: sometimes multiple artists or bands are contained in one label, and we expect links to DBpedia for all of them.

Selecting the optimal c_2 and s_2 values We apply Spotlight's annotate function on the output of the exact match phase for the Last.fm Artist dataset with the best performing optimization approach (precision-based optimization). The parameters that can be set are confidence and support, which we call c_2 and s_2 to differentiate them from the same parameters of the exact match phase. Only the best $k = 1$ candidate is considered, as we can now be significantly less certain of the validity of matches. We again cannot optimize these parameters in an unsupervised way, so we use the reference alignment again to optimize in terms of F_1, analogously to finding the optimal baseline in section 4.2.1. The hillclimbing graph is shown in figure 4.4. The optimal results are obtained when c_2 is set to 0.63 and s_2 set to 25, with $Precision = 0.95$, $Recall = 0.903$ and $F_1 = 0.930$, which is an improvement of 0.5% in terms of absolute F_1-score compared to the result of the exact match phase, which was 0.925. So there is merit in applying an additional partial match phase if it is required to match entities hidden within labels as well. It is difficult to say whether these parameter values are also good defaults, as they are most likely to be different for different datasets. It does seem clear that high thresholds will yield the best results.

4.4.4 Effect of number of collected classes/categories on performance

We check how the result is affected if the number of DBpedia resources from which classes and categories are collected is altered. Changing this can affect which classes end up in the collection after the various pruning steps. Up until now, this has been kept constant at 100 resources, but it would be interesting to know if results improve given a larger sample of classes/categories (and thus more reliable domain derivation), or how low we can go before results start to deteriorate.

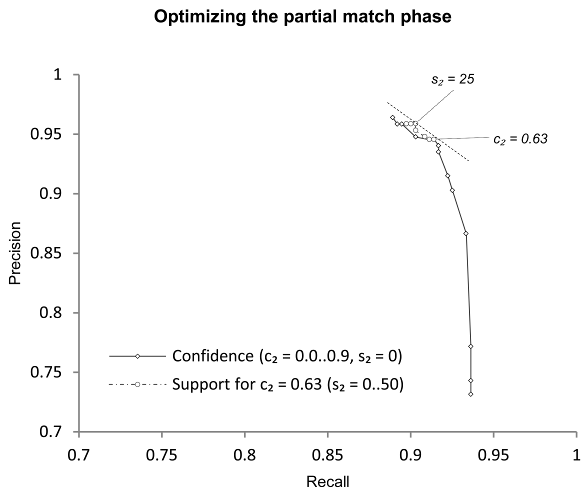


Figure 4.4: Hillclimbing during the partial match phase to optimize the F_1-score for the Last.fm use case. The point marked by the diagonal corresponds to the optimal F_1-score ($F_1 = 0.930$).

Table 4.8: Results for the precision-based optimization approach on Last.fm Artists for different class/category collection sizes. The previously assumed size of 100 is displayed in bold. The resulting domains for the remaining sizes are displayed in terms of difference from this reference domain.

Size	*Precision*	*Recall*	F_1	Domain
20	0.964	0.889	0.925	+YAGO: 1990sMusicGroups
50	0.964	0.889	0.925	+Category: Musical_groups_by_nationality
100	**0.964**	**0.889**	**0.925**	DBpedia: Band, YAGO: MusicalOrganization, YAGO: 2000sMusicGroups, Category: Musicians, Category: Musicians_by_nationality, Category: Musicians_by_genre, Freebase: /music/musical_group, Freebase: /music/group_member, Freebase: /music/artist
200	0.964	0.889	0.925	+Category: Musical_groups_by_nationality
500	0.964	0.889	0.925	+Category: Musical_groups_by_nationality

This is again tested for the best performing approach on the Last.fm Artist dataset, precision-based optimization. We check results for 20, 50, 100, 200 or 500 DBpedia resources from which to collect classes and categories from. The results are displayed in table 4.8. Although there are some differences in the resulting domains, precision, recall and F_1-score measures remain the same. Class/category collection size is thus not a very important factor to influence the final result, and even a very small sample size already produces a good initial domain.

4.5 Example application: ImREAL simulators

The EU ImREAL project[2] is intended to promote research into the creation of a seamless link between a simulated learning experience and real-world job-experiences. We focus on psychiatric role-playing simulators provided as a use case for the project. They work as follows: a user takes on the role of the psychiatrist and must apply his expertise in handling a patient with a mental disorder. A run through a simulator consists of a sequence of choices that the user can make. The choices themselves represent the possible answers that the psychiatrist can give. For each choice a different score is assigned. At the end of the run, the user is scored by the accumulation of this score.

These simulators are meant to be adaptive, so that they can be steered to ask a user more about topics he scores low on, and provide him additional information and pointers concerning these topics. To facilitate this, the simulated environment needs be augmented with more information. We can use our approach to enrich these simulators with additional information from DBpedia and the Linked Data cloud.

The first step is to analyze the simulator definitions, which are written in standard XML, and adapt them to make them interoperable with the Semantic Web, which greatly facilitates the data integration that we are seeking. We developed an ontological model for the simulator and converted the XML simulator definitions to RDF. This process is described in appendix C. Once we have the ontology, we will focus on enriching symptoms associated to patient dialogue transcripts. These were originally short sentences delimited with square brackets in the transcripts (e.g. "anxieties about medication," "early morning lack of motivation"); we extracted them into a separate RDF property `inmed:hasSymptom` of each `DialogElement` instance. There are 139 unique instances of these.

For this scenario, we can apply our approach as-is for each symptom sentence, providing the dialogue transcripts they are associated to as context. However, since there are only 139 symptom instances with the actual terms often hidden within the sentence, just an exact match phase trying to match only the full labels will not provide enough results to derive a domain filter. So here the partial matcher with the annotate function needs to be used for the bootstrap process as well. Because of this, we can expect lower quality performance over pure ontology matching cases where matching a full label is usually possible, as there is a second problem involved: a term needs to be identified before it can be disambiguated.

4.5.1 Results

The `inmed:hasSymptom` and `inmed:responseTranscript` properties containing the labels and context are fed to the system. An initial baseline exact + partial match phase without setting a confidence or support threshold identified 268 matches. We choose to generate a bootstrap linkset of around 50 positive and negative matches for these, so $R'_{50/50}$. This is achieved for a bootstrap with confidence thresholds of 0.6 resp. 0.25 for collecting positive and negative matches. The overall best-performing precision-based optimization (PBO) is applied and the result is compared with a manually composed reference alignment of the entire set of 139 symptom sentences (each sentence can

[2]urlhttp://www.imreal-project.eu/

Table 4.9: Results for matching symptom sentences of the ImREAL simulators.

Approach	c	s	c_2	s_2	*Precision*	*Recall*	F_1
PBO	0	0	0	0	0.927	0.650	0.764
Optimized baseline	0	0	0.17	77	0.596	0.657	0.625
True baseline	0	0	0	0	0.354	0.839	0.498

have multiple matches to DBpedia). So here, we essentially try to match each of the 268 terms identified with Spotlight's annotate function with thresholds at 0.

To evaluate this approach, we compare with true and optimized baselines. For the true baseline, confidence c, c_2 and support s, s_2 are kept at 0 like the domain-restricted approach and for the optimized baseline a theoretical upper bound is sought by optimizing c, c_2, s and s_2.

The results are displayed in table 4.9. We can see that a domain-restricted approach gives a significantly better result over both baselines (it outperforms the true baseline by 26.6% and the optimized baseline by 13.9%), even with a very limited sample set of bootstrap links available. This is a different problem from pure ontology matching, as it is closer to free text matching than term matching. While the absolute quality of matching from free text is inevitably much lower, the result is promising and implies that, if properly adapted, the domain-aware matching approach can provide a significant improvement to the annotation of free text as well.

4.6 Conclusion and discussion

This chapter described the evaluation for the domain-aware matching approach. It was shown that when the dataset we want to match to DBpedia is particularly ambiguous, such as the case for Last.fm artists, a tremendous gain in matching quality can be obtained by first bootstrapping these matches and deriving a domain filter to restrict them. With the best-performing approach, a 17.1% improvement in F_1-score was gained over a baseline, domain-unaware approach. For the UMLS "Physiologic Function" subdomain, which is a much more specialized dataset but still contains some ambiguous terms, we still saw an improvement of 4.4%. On the other hand, given an extremely specialized and unambiguous dataset such as the "Pathologic Function" subdomain of the UMLS, matching with a domain filter does not provide a significant improvement, and can potentially even hurt quality. Still, an improvement over the baseline was gained once we derived a domain filter based on the entities of a super-class of Pathologic Function, "Phenomenon or Process": all optimization approaches consistently outperformed the baseline, with the largest improvement being 1.1%.

The overall results show that relying solely on the f_1-scores, choosing whichever method provided the highest value, may not always be the best option – the type of approach used to achieve the f_1 seems to play an important role as well. For all datasets, the precision-based optimization approach performs quite well, being the best performer on the Artist, Physiologic Function and Process or Phenomenon datasets and the second best performer on the Pathologic Function dataset. This despite the fact that it never attains the highest f_1 compared to the other approaches. It is also the fastest optimization approach, requiring at most 80 alignment instance generations.

The genetic algorithm is the most reliable in obtaining the highest value for f_1. It does so for all datasets except for the Last.fm artists, where it is 0.1% lower than what the broadness-based method achieves. However, the genetic algorithm is very expensive in terms of computational complexity. Optima were shown to be obtained after 20 to 25 evolutions. With the settings we used this requires roughly 4000 alignment instance generations.

As can be observed from the tables of the precision-based optimization method (table 4.4), there is large variety in types of domain information (i.e. DBpedia, YAGO and Freebase classes and categories) that gives highest precision for the underlying datasets. Recall that when a single-class filter gives high precision, this means that there are only very few false positive matches, meaning that this class effectively filters out matches that are not relevant to the domain. This justifies the choice to collect domain information from many different sources, as it shows that each dataset has different types that are most relevant. For Last.fm, we see that classes of every type are very useful, and categories less so. For the Pathological Function class, domain knowledge can apparently be most reliably extracted from the YAGO hierarchy. For the Physiological Function class, categories are now the most important.

It was shown that deriving a domain filter from the lowest possible sub-domains in an ontology or dataset is not always the best option. When evaluating against the reference alignment of UMLS class Pathologic Function, matching with a domain filter derived from super-class Phenomenon or Process outperforms matching with a domain filter derived from the original class. A domain that is too specialized will yield an equally specialized domain filter; it depends on the domain whether DBpedia contains these types of specialized classes/categories for *all* corresponding resources. Furthermore, when entities are inherently very unambiguous, the gain from a domain filter becomes negligible. For future improvements to the approach, we need to be able to detect when a certain sub-class is too specialized and unambiguous, so that the domain of a super-class can be used instead to improve recall and F_1 for the matches for the sub-domain.

We showed that it is best to consider the best-2 ranked candidate results when using a domain filter. For further matching improvements, a second partial match phase may be applied, but whether this is desirable depends on the application. With high annotation threshold values, a second matching pass can give an improvement over the exact match phase, but it is unclear which exact values will be most suitable for which dataset. Lastly, the sample size of DBpedia resources from which to collect classes and categories from during the bootstrap phase has been shown to have very little impact on the final result.

For applications where an exact match phase is not useable, such as for the matching of free text, applying the full approach based on partial matches can yield significant improvements over baseline approaches, although the absolute quality is lower than when exact matching is possible. The approach that we have developed has focused largely on the scenario of ontology matching, where one or more full labels can be considered for the matching process. The ImREAL simulator application showed that it could work quite well for free text annotation as well, but more research is needed into how the approach can be adapted for this.

Chapter 5

Conclusions and Future Work

This chapter concludes the thesis with a summary of the performed work and obtained results (5.1), answers to the research questions (5.2) and a listing of possible directions for future work (5.3).

5.1 Summary

This thesis presented work on the development and evaluation of a domain-aware ontology matching approach. To facilitate data integration on the Semantic Web, concepts in the Linked Data cloud need to be interlinked. At present, the cloud is quite clustered, with far too few links between identical concepts of ontologies from different domains. This lack of interconnectivity can be alleviated by linking these domain-specific ontologies to a single, cross-domain ontology such as DBpedia. Creating these links enriches the domain-specific data with more general data. On top of this, DBpedia can act as an intermediary to interlink two domain-specific ontologies.

Over the course of this work, we showed that we can vastly improve the quality of generated links over traditional approaches by bootstrapping the matching process – we derive a filter that describes and delimits the domain of the source dataset in terms of classes and categories collected from DBpedia.

The domain derivation is accomplished in a fully unsupervised way. First, a sample of high-confidence links from the entities of a domain-specific dataset to DBpedia resources is generated, using context similarity measures. From these DBpedia resources, classes and categories are collected. This collection of classes and categories is then pruned and optimized to obtain a precise encapsulation of the domain of interest. Optimization can be accomplished by testing the quality of domain filters on the sample of high-confidence links – we showed that a correlation exists between performance according to this bootstrap linkset and performance on the actual data.

Several different methods for pruning and optimizing the domain were presented and evaluated for two use cases: Last.fm artists and UMLS medical terms. It was shown that for ambiguous data such as artist and band names, the proposed approach outperformed a traditional matching approach by as much as 17.1% on an F_1 scale of quality. Improvements for highly specialized data such as medical terms for diseases were less striking, but could still be gained by taking a super-class of the class of terms as domain of reference, to avoid overfitting the domain filter. Finally, given an example application of medical training simulators, it was shown that the domain-aware approach is also potentially very useful when adapted to the annotation of free text, where a gain of as much as 26.6% was attained over traditional annotation approaches.

5.2 Answers to the research questions

We will reiterate the research questions that we started out with and summarize the answers that were obtained over the course of this thesis.

1. ***How can we exploit the domain of a domain-specific ontology to improve alignment quality to a general-purpose ontology?***
   This was the main theme of the thesis, thus the entirety of chapter 3 is the answer to this question. Refer to the summary above for a compact version. We showed that we can collect domain information from DBpedia and effectively use this as a domain filter for matching.

2. *To what extent can such a domain derivation process be automated?*
   We showed that there is a strong correlation between performance according to an automatically generated bootstrap linkset and performance on the actual dataset. Since we can optimize on a quality measure calculated with the bootstrap linkset, we can derive high-quality domain filters in a fully unsupervised way. The only input required from the user is a source dataset and a definition of which class of entities to match, which property or properties to use as label(s) for matching and which property or properties to use as context.

3. *What is the impact on alignment quality when ontology domains are taken into consideration?*
   In most cases, matching scores for the domain-aware approach were shown to be significantly better than for domain-unaware baselines. For highly specialized and unambiguous data, it can happen that domain filters do not lead to a significant improvement in alignment quality. However, fundamentally, the more ambiguous the source entities, the higher the difficulty in matching these to corresponding entities of the correct senses. And it is for these most difficult cases that the positive impact on quality is most significant when using the domain-aware approach. So while the approach will not yield benefits for every single case, the results are nonetheless highly encouraging.

   a) *How is this impact affected when considering ontologies of different and/or broader domains?*
      The type of domain was shown to be irrelevant – we showed outstanding performance on the Last.fm Artist dataset, and saw an improvement on medical datasets as well. The improvement was minor on highly specialized medical terminology, but very significant for the symptoms in a less specialized training simulator environment. This showed that it is mainly the ambiguity of data that has the highest impact on alignment quality. We also showed that deriving a domain filter from a broader domain, i.e. a super-class of a class of entities, can lead to better matching for a sub-class of entities if this sub-class is highly unambiguous. Overall performance

impact when matching the entire broader domain was as expected – the performance gets averaged over the performance obtained when matching with domain filters for the individual sub-classes separately.

4. *How can real-world use cases benefit from this approach?*
   We showed that the ImREAL medical simulators can benefit from the approach when matching symptoms to DBpedia, as it was shown that is also highly effective for annotating the type of free text prevalent in these simulators – we achieved very high precision with next to no compromise in recall compared to standard annotation approaches.

5.3 Future work

There are a number of points where future research could lead to improvements of the current method, or to additional insights into the issues we dealt with.

Generalization Our method is currently fixed to DBpedia as target ontology, but could theoretically work in a generic ontology matching system as well. Future work could be to generalize the current system to also support different target ontologies, or to integrate our approach as a separate step into already existing ontology matching systems.

Adaptation and evaluation for free text This work was focused around an ontology matching scenario, which is characterized by well-defined entities which usually have labels that concisely describe them. These labels can then generally be used as-is to find direct or indirect correspondences, so that the task is reduced to the disambiguation of entities. When dealing with regular text, we are faced with the additional task of identifying named entities before they can be disambiguated. We showed for the ImREAL example application in section 4.5 that significant gain can be achieved with our method for the annotation of domain-specific text as well. However, more work is needed into adapting the approach and exploring what works best for domain-restricted matching in a free text scenario.

Discover when to use a more generic domain In section 4.4.1 of the evaluation, we showed that for the UMLS Pathologic Function class individually, better results were obtained when deriving the domain from a super-class instead. But for the same experiment performed for the Physiologic Function class, results were shown to be worse. We conjectured that the reason for the improved performance was the fact that the Pathologic Function class is too unambiguous, and therefore a domain restriction on just this class quickly gets too tight, excluding potentially correct matches from the result. We need to find out a way to effectively determine when a class is extremely unambiguous, so that a broader class can instead be used to derive a less restrictive domain filter.

Custom candidate index We used the DBpedia Spotlight candidate index – which is pre-generated from Wikipedia to facilitate the discovery of correspondences by maintaining a list of candidate resources per surface form – without any modifications. However, we observed that whenever links that should be present according to the reference alignment are missing, the majority of the time this is due to the fact that the surface form(s) on which we match simply do not exist in Spotlight's candidate index. This is partly because the index is optimized for the annotation of free text, and is stripped of surface forms that consist only of common words. However, especially in the case of Last.fm artists, it happens often that a band name is composed of only common words (e.g. "Hey" or "Six by Seven," both band names contained in our reference alignment as well as in DBpedia). These entities are never properly matched. Less common, but still prevalent, are cases where a surface form points to a disambiguation page, and this disambiguation page contains a link to the resource we need, but is still somehow not included in the index. This points to parsing issues that arise for certain pages, causing the indexer to miss these candidates.

Creating a custom parser that is specialized towards ontology matching could significantly improve the absolute matching scores for all of the use cases. Ontology matching is a specialized application field where it is unneeded to filter surface forms composed of common words. Improving the general accuracy of the parser itself should also lead to better results.

Evaluation with other ontology matching systems A great number of ontology matching systems have been developed. Most are generic, intended to align any source and target ontology given to it. There are no other ontology matching approaches that focus on matching arbitrary source ontologies to DBpedia that we know of. Comparison of our approach *against* state-of-the-art ontology matching systems is therefore not informative, as it would give us an unfair advantage. Evaluating the approach *with* a generic ontology matching system *is* possible, by integrating the derivation of a domain filter as a pre-processing step for the matching, and comparing performance between standard operation and domain-restricted operation. In that sense, it is an extension to the first future work discussed: generalization.

Bibliography

[1] Burgun A and Bodenreider O. Comparing terms, concepts and semantic classes in WordNet and the Unified Medical Language System. In *Proceedings of the NAACL 2001 Workshop: WordNet and other lexical resources: Applications, extensions and customizations*, pages 77–82, 2001.

[2] Ontotext AD. Linked life data, a semantic data integration platform for the biomedical domain, Retrieved on December 14 2011.

[3] P.L. Araúz, P. Faber, and P.J.M. Redondo. Linking domain-specific knowledge to encyclopedic knowledge: an initial approach to linked data. In *Proceedings of the 2nd International Workshop on the Multilingual Semantic Web at ISWC 2011*, page 68, 2011.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science*, chapter 52, pages 722–735. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.

[5] D. Beckett. Turtle - terse rdf triple language, Retrieved on December 14 2011.

[6] V. Richard Benjamins, Jesús Contreras, Oscar Corcho, and Asunción Gómez-pérez. Six Challenges for the Semantic Web. In *In KR2002 Semantic Web Workshop*, volume 1, 2002.

[7] T. Berners-Lee. Primer: Getting into rdf and semantic web using n3, Retrieved on December 14 2011.

[8] Tim Berners-Lee. Cool uris don't change, Retrieved on December 14 2008.

[9] Tim Berners-Lee and Mark Fischetti. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, September 1999.

[10] Tim Berners-Lee, J. Hendler, and O. Lassila. The Semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 5(1), 2001.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, MarMar 2009.

[12] D. Brickley and L. Miller. Foaf vocabulary specification 0.98, Retrieved on December 14 2011.

[13] J. Brooke. Sus – a quick and dirty usability scale., 1996.

[14] M. Dean, G. Schreiber, S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L. Andrea Stein. Owl web ontology language reference w3c recommendation, Retrieved on December 14 2011.

[15] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 1st edition, July 2007.

[16] Christophe Guéret. Exploring linked data content through network analysis. talk at yahoo! seminar, November 23 2011.

[17] Willem Robert Van Hage, Antoine Isaac, and Zharko Aleksovski. Sample evaluation of ontology-matching systems. In *Fifth International Workshop on Evaluation of Ontologies and Ontology-based Tools, ISWC 2007, Busan, Korea*. Springer, Heidelberg, 2007.

[18] Sebastian Hellmann, Claus Stadler, Jens Lehmann, and Sören Auer. DBpedia Live Extraction. In *OTM '09: Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems*, pages 1209–1223, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] IBM. Watson, Retrieved on December 14 2011.

[20] Nancy Ide and Jean Véronis. Introduction to the special issue on word sense disambiguation: the state of the art. *Comput. Linguist.*, 24:2–40, March 1998.

[21] Antoine Isaac and Ed Summers. Skos simple knowledge organization system primer, Retrieved on December 14 2011.

[22] Robert Isele and Christian Bizer. Learning linkage rules using genetic programming. In *Proceedings of the Sixth International Workshop on Ontology Matching at ISWC 2011*, volume 814, page 13. CEUR-WS, 2011.

[23] Prateek Jain, Pascal Hitzler, Amit P. Sheth, Kunal Verma, and Peter Z. Yeh. Ontology alignment for linked open data. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 402–417, Berlin, Heidelberg, 2010. Springer-Verlag.

[24] Prateek Jain, Peter Yeh, Kunal Verma, Reymonrod Vasquez, Mariana Damova, Pascal Hitzler, and Amit Sheth. Contextual ontology alignment of lod with an upper ontology: A case study with proton. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and

Jeff Pan, editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 80–92. Springer Berlin / Heidelberg, 2011.

[25] Aditya Kalyanpur, J. William Murdock, James Fan, and Christopher Welty. Leveraging community-built knowledge for type coercion in question answering. In *Proceedings of the 10th international conference on The semantic web - Volume Part II*, ISWC'11, pages 144–156, Berlin, Heidelberg, 2011. Springer-Verlag.

[26] H. Khrouf and R. Troncy. Eventmedia live: Reconciliating events descriptions in the web of data. *6th International Workshop on Ontology Matching, Bonn, Germany*, 2011.

[27] Georgi Kobilarov, Tom Scott, Yves Raimond, Silver Oliver, Chris Sizemore, Michael Smethurst, Christian Bizer, and Robert Lee. Media meets semantic web – how the bbc uses dbpedia and linked data to make connections. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl, editors, *The Semantic Web: Research and Applications*, volume 5554 of *Lecture Notes in Computer Science*, pages 723–737. Springer Berlin / Heidelberg, 2009.

[28] C. Lindberg. The Unified Medical Language System (UMLS) of the National Library of Medicine. *Journal (American Medical Record Association)*, 61(5):40–42, May 1990.

[29] C. E. Lipscomb. Medical Subject Headings (MeSH). *Bulletin of the Medical Library Association*, 88(3):265–266, July 2000.

[30] Sean Luke. *Essentials of Metaheuristics*. lulu.com, March 2011.

[31] D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res*, 33(Database issue), January 2005.

[32] Frank Manola and Eric Miller. Rdf primer, Retrieved on December 14 2011.

[33] Deborah L. Mcguinness and Frank van Harmelen. OWL Web Ontology Language Overview.

[34] Edgar Meij, Marc Bron, Laura Hollink, Bouke Huurnink, and Maarten de Rijke. Mapping queries to the Linking Open Data cloud: A case study using DBpedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, April 2011.

[35] Pablo N. Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics)*, 2011.

[36] George A. Miller. WordNet: a lexical database for English. *Commun. ACM*, 38(11):39–41, November 1995.

[37] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Hoeffner. RAVEN – Active Learning of Link Specifications. In *Proceedings of the 6th International Workshop on Ontology Matching (OM-2011)*, volume 5554, page 25. CEUR-WS, 2011.

[38] U.S. National Library of Medicine. Current semantic types, Retrieved on December 14 2011.

[39] U.S. National Library of Medicine. Pubmed, Retrieved on December 14 2011.

[40] U.S. National Library of Medicine. Snomed clinical terms, Retrieved on December 14 2011.

[41] U.S. National Library of Medicine. The umls semantic network, Retrieved on December 14 2011.

[42] U.S. National Library of Medicine. Unified medical language system, Retrieved on December 14 2011.

[43] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf, Retrieved on December 14 2011.

[44] D. Ritze and H. Paulheim. Towards an automatic parameterization of ontology matching tools based on example mappings. In *Proceedings of the Sixth International Workshop on Ontology Matching at ISWC 2011*, volume 814, page 37. CEUR-WS, 2011.

[45] G. Rizzo and R. Troncy. Nerd: Evaluating named entity recognition tools in the web of data. In *ISWC'11, Workshop on Web Scale Knowledge Extraction (WEKEX'11),October 23-27, 2011, Bonn, Germany*, 2011.

[46] Leo Sauermann and Richard Cyganiak. Cool URIs for the semantic web. Technical Report W3C Interest Group Note 31 March 2008, W3C, 2008.

[47] Pavel Shvaiko and Jérôme Euzenat. Ten challenges for ontology matching. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*, OTM '08, pages 1164–1182, Berlin, Heidelberg, 2008. Springer-Verlag.

[48] Kristian Slabbekoorn, Laura Hollink, and Geert-Jan Houben. Domain-aware matching of events to dbpedia. In *Proceedings of the Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE 2011), in conjunction with the 10th International Semantic Web Conference 2011 (ISWC 2011), Bonn, Germany*, volume 779. CEUR-WS, 2011.

[49] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 697–706, New York, NY, USA, 2007. ACM.

[50] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin core metadata for resource discovery. retrieved december 14, 2011, September 1998.

[51] Gerhard Weikum. For a few triples more. keynote at the 10th international semantic web conference, October 27 2011.

[52] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, April 1997.

[53] Cathy H. Wu, Rolf Apweiler, Amos Bairoch, Darren A. Natale, Winona C. Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, Maria J. Martin, Raja Mazumder, Claire O'Donovan, Nicole Redaschi, and Baris Suzek. The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Research*, 34(suppl 1):D187–D191, January 2006.

Appendix A

Glossary

This appendix provides a glossary of commonly used terms and acronyms in the Semantic Web research field. Some of the terms listed are inherently somewhat vague, with no real common agreement on their formal meaning within the Semantic Web community. This list is intended to provide the reader explicit definitions of the senses in which these terms are used throughout this thesis.

Alignment: See **ontology alignment**.

Alignment description: In the context of this thesis, this is the document that describes the links of an alignment in $\langle label, URI \rangle$ pairs.

Class: Also called "type." A class defines a group of entities that belong together because they share some properties. Each entity is classified according to at least one class.

Context: Context is the labels and additional definitions that describe an entity. When determining whether two entities represent the same concept, the context of both is compared, and a similarity score is calculated.

Dataset: Used interchangeably with **ontology**.

Domain: In the context of this thesis, a domain is simply a union of one or more classes.

Entity: In the context of this thesis, an entity refers to an RDF resource that represents a concept. It is an instance of a class, therefore used interchangeably with "instance."

Individual: See **entity**. "Individual" is the official RDF naming for an entity, but in practice not often used.

Instance: See **entity**.

Link: The RDF connection made between matching entities (popularly done with OWL property `owl:sameAs`; in this work we prefer the SKOS property `skos:exactMatch`).

Match: See **positive match**.

Negative match: An entity in a source ontology has no corresponding entity in a target ontology.

Ontology: An ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain. In the context of the Semantic Web, ontology is often used interchangeably with **dataset**.

Ontology alignment: The output of ontology matching, i.e. a specification that says for each source entity which target entities it matches to (if any).

Ontology matching: The process of finding for each entity in a source ontology, the entity or entities in a target ontology that refer to the same concept.

OWL: Web Ontology Language. The schema language that describes properties of and relationships between entities within ontologies on the Semantic Web.

Positive match: A correspondence between an entity in a source ontology and an entity in a target ontology.

RDF: Resource Description Framework. The language for describing concepts and ontologies on the Semantic Web.

Resource: In the context of RDF, a resource refers to a class, property or instance. In the context of DBpedia, a resource is a DBpedia entity (i.e. the Linked Data form of a Wikipedia page).

Surface form: Used in the context of DBpedia Spotlight. It refers to a chunk of text that could be a mention of a DBpedia resource. The chunk of text is contained in Spotlight's candidate index with one or more DBpedia resources associated to it.

Appendix B

UMLS Semantic Network Hierarchy

This appendix lists the full hierarchy of `is-a` inter-class relations of the UMLS Semantic Network [38]. It is shown on the next page. The hierarchy has two roots: "Entity" and "Event." Note that the top part of the second column is a continuation of the first column, with "Conceptual Entity" a direct subclass of "Entity." Highlighted is the part that is used in this work. The numbers in bold refer to the sizes of each class, i.e. the number of instances this class and all of its sub-classes contain.

- Entity
  - Physical Object
    - Organism
      - Virus
      - Bacterium
      - Archaeon
      - Eukaryote
        - Plant
        - Fungus
        - Animal
          - Vertebrate
            - Amphibian
            - Bird
            - Fish
            - Reptile
            - Mammal
              - Human
    - Anatomical Structure
      - Embryonic Structure
      - Anatomical Abnormality
        - Congenital Abnormality
        - Acquired Abnormality
      - Fully Formed Anatomical Structure
        - Body Part, Organ, or Organ Component
        - Tissue
        - Cell
        - Cell Component
        - Gene or Genome
    - Manufactured Object
      - Medical Device
        - Drug Delivery Device
      - Research Device
      - Clinical Drug
    - Substance
      - Chemical
        - Chemical Viewed Functionally
          - Pharmacologic Substance
            - Antibiotic
          - Biomedical or Dental Material
          - Biologically Active Substance
            - Neuroreactive Substance or Biogenic Amine
            - Hormone
            - Enzyme
            - Vitamin
            - Immunologic Factor
            - Receptor
          - Indicator, Reagent, or Diagnostic Acid
          - Hazardous or Poisonous Substance
        - Chemical Viewed Structurally
          - Organic Chemical
            - Nucleic Acid, Nucleoside, or Nucleotide
            - Organophosphorus Compound
            - Amino Acid, Peptide, or Protein
            - Carbohydrate
            - Lipid
              - Steroid
              - Eicosanoid
          - Inorganic Chemical
          - Element, Ion, or Isotope
      - Body Substance
      - Food
  - Conceptual Entity
    - Idea or Concept
      - Temporal Concept
      - Qualitative Concept
      - Quantitative Concept
      - Functional Concept
        - Body System
      - Spatial Concept
        - Body Space or Junction
        - Body Location or Region
        - Molecular Sequence
          - Nucleotide Sequence
          - Amino Acid Sequence
          - Carbohydrate Sequence
        - Geographic Area
    - Finding
      - Laboratory or Test Result
      - Sign or Symptom
    - Organism Attribute
      - Clinical Attribute
    - Intellectual Product
      - Classification
      - Regulation or Law
    - Language
    - Occupation or Discipline
      - Biomedical Occupation or Discipline
    - Organization
      - Health Care Related Organization
      - Professional Society
      - Self-help or Relief Organization
    - Group Attribute
    - Group
      - Professional or Occupational Group
      - Population Group
      - Family Group
      - Age Group
      - Patient or Disabled Group

- Event
  - Activity
    - Behavior
      - Social Behavior
      - Individual Behavior
    - Daily or Recreational Activity
    - Occupational Activity
      - Health Care Activity
        - Laboratory Procedure
        - Diagnostic Procedure
        - Therapeutic or Preventive Procedure
      - Research Activity
        - Molecular Biology Research Technique
      - Governmental or Regulatory Activity
      - Educational Activity
    - Machine Activity
  - Phenomenon or Process **198665**
    - Human-caused Phenomenon or Process **531**
      - Environmental Effect of Humans **52**
    - Natural Phenomenon or Process **126354**
      - Biologic Function **125717**
        - Physiologic Function **54781**
          - Organism Function **4872**
            - Mental Process **743**
          - Organ or Tissue Function **3279**
          - Cell Function **10804**
          - Molecular Function **34483**
            - Genetic Function **3331**
        - Pathologic Function **70046**
          - Disease or Syndrome **55989**
            - Mental or Behavioral Dysfunction **2943**
            - Neoplastic Process **13245**
          - Cell or Molecular Dysfunction **1089**
          - Experimental Model of Disease **71**
    - Injury or Poisoning **70497**

Appendix C

Development of an ImREAL simulator ontology

This appendix provides a description of the work that was done to convert the provided ImREAL training simulators into RDF. It was developed using the psychological simulator use case, but is generic enough to describe simulators of other domains as well. This conversion is intended to simplify enrichment and data integration with external Linked Datasets such as DBpedia and Linked Life Data.

C.1 Domain model

The first step in the development of an enrichment process is to model the domain of the training simulator that we need to augment. In the use case presented to us, this training simulator consists of interviews in the psychiatric domain, in the form of role-playing simulators. They are targeted to medical students that are still inexperienced and have conducted little to no actual interviews with patients yet. The goal of the simulators is to train these students in patient interviews – to give them the confidence and knowledge to be able to adequately perform interviews with live persons.

The creation of a domain model can be split up into two phases. First, the training simulator themselves and the logs they produce need to be analyzed, so that we know which data is available to us, and which data is relevant to extract. Second, all relevant information needs to be modeled in a standard format in a meaningful way. In our case, since we want our model to be interoperable with the Linked Data cloud, this involves the creation of an ontology in RDF/OWL format. A well-defined domain model, containing all data we have pertaining to the simulators, serves a number of purposes:

- Interoperability of data: by having all data defined using Linked Data standards, it becomes easy to integrate it with other data.

- Availability of data: by having as much data available as possible, new data could be inferred by combining it with other local or remote data sources at any time.

- Improved provenance: by having the source data available, the pieces of information that have been inferred from several facts can be traced back to what they were derived from.
- Clarity: a clear model gives us valuable insight into and a better understanding of the workings of the system.

C.2 Analysis

Based on the information provided to us via presentations and log files, we can describe the general working principles of the simulators. A run through a simulator consists of a sequence of statement/response pairs. An interview starts with a fixed greeting statement and response from the patient. The user is shown patient responses through video clips. A user, in the role of the psychiatrist, is then allowed to pick a next statement from a number of multiple choice options. This statement can be anything ranging from questions to summarizing statements to advice. The response from the patient is fixed for each statement. This process of picking a statement and receiving a scripted response continues until the interview ends. At the end, the user is assigned a score based on the choices he made.

For our use case we have been given XML descriptions of two simulators, dealing with the topics of "depression" and "mania," and a collection of logs of users' runs through the simulators. We identify relevant information items from these resources. The simulators consist of the following:

- A simulator ID, name and description.
- A collection of categories (shared across simulators): these describe the type of statements that can be made (e.g. "closed question," "summarizing statement," etc.).
- A collection of subjects (shared across simulators): these describe the subject of a statement made (e.g. "appetite," "hallucinations," etc.).
- A collection of sections (particular to each simulator): these delineate the various phases of the interview (e.g. "introduction," "treatment plan," etc.).
- A collection of dialog elements. A dialog element consists of the following:
  - The transcript of the statement by the psychiatrist.
  - The transcript of the response to this statement by the patient.
  - The category of the statement.
  - The subject of the statement.
  - The symptom(s) that can be detected in the patient's response (listed between brackets at the end of the response).
  - The section the element belongs to.
  - Relationships with other dialog elements (i.e. the elements that follow it. These are essentially the choices available to the user).

  - A score between 0.0 and 1.0 assigned to each relationship. These are score multipliers.

The following is known about the users:

- An ID number. The simulator logs are anonymous, and nothing is known about the user's identity aside from this.
- Their runs through the simulator. For each run the following is known:
  - The start and end times for the run.
  - The total score.
  - The sequence and number of dialog elements traversed. We will henceforth refer to the user's traversal of a particular dialog element as a *dialog event*.
  - The start times for each dialog event.
  - The sections it comprises. Each section can effectively be considered a sub-run – (a) through (d) is also known for each individual section.
- Some properties derived from personal questionnaires answered by the user (e.g. the user's past experience with interviews, their motivation before and after using the simulator, etc.).
- A system usability scale (SUS) [13] questionnaire that details the user's experiences with the system. This seems to have been intended mostly for the designers of the simulators to improve the system based on this feedback, and is thus not very useful for our purposes.

The above needs to be captured in a model. There are several demands that this model should satisfy. The model should be

- comprehensive: all of the above identified information needs to be included.
- intuitive: choices of ontology terms and relations and properties used should be logical, so that the data will remain simple to query.
- compatible: existing ontologies should be reused wherever possible.

C.3 Modeling

Now that we have made an inventory of the relevant pieces of information known to us, we need to devise an OWL [14] ontology to represent the domain in a coherent way, while sticking to common RDF design principles such as those described in the W3C RDF primer [32] and cool URIs [8][46]. The ontology consists of two things: a schema which the data should adhere to, and the data itself.

A graphical representation of the model we devised is shown in figure C.1. In this particular representation, the focus is on the relations between classes and resources; datatype properties have been omitted. We have chosen `inmed` ("interview medical

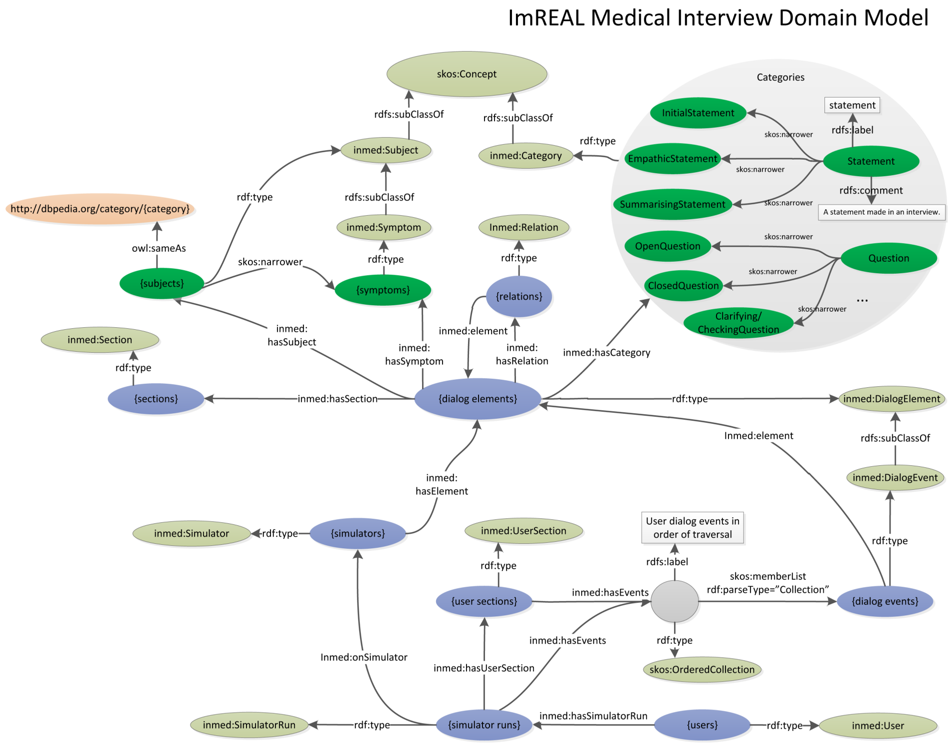


Figure C.1: Ontology for the ImREAL simulators. Classes are shown in yellow ovals, generic instances in blue ovals, example instances in green ovals and properties across arrows. The white rectangles are comments.

domain") as prefix for our ontology (`http://imreal-project.eu/ontology`). Resources will be stored in the `http://imreal-project.eu/resource/` base namespace and subfolders. Following below is a textual explanation of the model.

Simulators are instances of class `inmed:Simulator`, and each simulator has a list of `inmed:DialogElement` instances. This is represented as a list of `inmed:hasElement` properties covering all dialog elements contained in a simulator. The dialog elements need not be ordered, although they do contain relationships to other dialog elements (corresponding to the choices a user can make to advance the simulator). The relationships are modeled with the `inmed:followedBy` property, with a `DialogElement` as `owl:domain` and `owl:range`.

Each dialog element has a section, a subject, a category, and contains one or more symptoms in its response transcript. Sections are modeled as a class `inmed:Section`. Subjects, categories and symptoms are modeled as (subclasses of) `skos:Concept` instances. The concepts will be the most prominent target for domain knowledge enrichment. Having each concept handled within the simulators as instances of a commonly used vocabulary term such as SKOS [21] will serve to facilitate interoperability. As an example shown in the diagram, subject "Anxiety" might get a link to the DBpedia resource of the same concept. Additionally, the symptoms can be considered a subclass of `inmed:Subject`, as "Symptoms" itself is a subject and the symptoms in the transcript are essentially narrower related concepts of this. Subjects and categories are a pre-defined set of instances shared across all simulators, and can be placed in a concept hierarchy. The diagram shows a few examples.

Each user is defined as an `inmed:User` instance, and has a number of simulator runs of class `inmed:SimulatorRun`. Each simulator run is composed of an ordered set of dialog events, denoting the sequence of simulator elements traversed in a run. This set has been represented as a `skos:OrderedCollection` with the dialog events (of class `inmed:DialogEvent`) as objects of `skos:memberList` properties. Per the specification of `skos:OrderedCollection`, members of the `skos:memberList` are an RDF linked list in the correct ordering (it is up to the application to retain this order)[1]. Furthermore, `inmed:DialogEvent` is a subclass of `inmed:DialogElement`, as it is a more specific version of it; it is essentially a dialog element, but within a specified time frame in which it has been traversed. It has the corresponding dialog element as the object of property `inmed:element`.

Lastly, simulator runs consist of several sections, classified as instances of `inmed:UserSection`. This class is a subclass of `inmed:Section` – it is a more specific version of it, with a duration and score assigned, as well as references to the (ordered) dialog elements traversed in this section. It is also a subclass of `inmed:SimulatorRun`, as it is essentially a sub-simulator run: all user sections combined make the full simulator run.

[1] `http://www.w3.org/TR/swbp-skos-core-guide`