Internship Report — Mpri M2
Advances in Holistic Ontology Alignment

Antoine Amarilli, supervised by Pierre Senellart, Télécom ParisTech

General Context

The development of the semantic Web has given birth to a large number of data sources (*ontologies)* with independent data and schemas. These ontologies are usually generated from existing relational databases or extracted from semi-structured data. Ontology alignment is a technique to automatically integrate them by discovering overlap in their instances and similarities in their schemas. Such integration makes it possible to access the combined information of all these data sources simultaneously rather than separately. Ontology alignment must deal with numerous challenges: the schemas are heterogeneous, the volume of data is huge, and the information can be partially incomplete or inconsistent.

Problem Studied

My internship focuses on the Paris (Probabilistic Alignment of Relations, Instances, and Schema) ontology alignment system [SAS11] developed by Fabian Suchanek, Pierre Senellart, and Serge Abiteboul at Inria Saclay's Webdam team. Paris is a generic system which aligns data and schema simultaneously (hence the adjective "holistic") and uses each of these alignments to cross-fertilize the other. The alignments produced are annotated with a confidence score and have a probabilistic interpretation. Paris was able to align large real-world datasets from the semantic Web.

The aim of my internship is to propose improvements to Paris on some problems (both theoretical and practical) that were left open by the original system: achieving a better understanding of the behavior of Paris's equations, aligning heterogeneous schemas, being tolerant to differences in the strings of both ontologies, and improving the overall performance of the system.

Proposed Contributions

The contributions of my internship are the following: the implementation of performance improvements to Paris through parallelization, in-memory computation, and other techniques (section 3); the study and implementation of join relation support to align simple cases of heterogeneous schemas (section 4); a theoretical analysis of the behavior of Paris (section 5); support for approximate literal matching as a replacement for ad-hoc normalization techniques (section 6, joint work with Mayur Garg); and an application of Paris to deep Web analysis which makes use of these improvements (section 7, joint work with Marilena Oita).

The result of my implementation efforts was released on the Paris website [SSG$^+$], and the deep Web analysis contribution was accepted as a vision paper to the VLDS workshop of VLDB [OAS12]. As an additional contribution, I have studied the problem of efficient DAG labeling schemes during a three-week visit at Fabian Suchanek's ontology group in Saarbrücken and wrote a summary of my efforts during this time [Ama], not further discussed in this report.

Arguments Supporting Their Validity

The new setup and improved implementation was shown to achieve a ten-fold speed improvement for one of the main alignment tasks used to benchmark the system. The support for join relations was found to be useful in small matching tasks (especially for crawl results from the deep Web) though it cannot be run on large ontologies for performance reasons. The theoretical analysis of Paris hints at a link between the probabilistic model justifying the equations of Paris and a variant of Green measures, which suggest possible alternative choices for the design of Paris. The literal matching technique was benchmarked and shown to match the performance of ad-hoc normalization used for one of the original datasets used to evaluate Paris. The application of Paris to deep Web analysis showed promising results by managing to outline the semantics of the Amazon book search Web form, which showcased the use of join relations and approximate literal matching.

Summary and Future Work

During my internship, I have proposed several improvements to the Paris system which address some of its original shortcomings. This contribution should both improve the performance of Paris on existing matching tasks and make it possible to attempt matching tasks between more heterogeneous or noisy ontologies. The next step would be to obtain more experimental proof of this claim.

Several questions are left open. The natural generalization of join relation alignment is the alignment of arbitrary patterns between ontologies, though we would need an elaborate way to chose candidate patterns to ensure that the task is still tractable. Approximate literal matching techniques could be extended to handle all datatypes and not just strings. Other possible incremental improvements to the Paris system would be an improved class alignment to align unions and intersections of classes, though this is another special case of generic pattern alignment.

More fundamental efforts to improve Paris would be to achieve a better understanding of its model by linking it to such formalisms as weighted Max-SAT or Markov Logic Networks [RD06], and maybe use these formalisms to derive a new model with stronger guarantees. Another ambitious goal would be to ensure the scalability of Paris to larger tasks, either through distributed computation or with a different, more efficient algorithm.

A very interesting “next question” would be the design of an ontology matching system inspired by Paris but scaled to a large number of data sources from the Web. The aim of such efforts would be to integrate both general and specialized data sources of various natures on the Web: semantic Web ontologies, automatically extracted information from semi-structured data or natural language text, and data crawled from the deep Web. Such a task would lead to deep questions on the management of uncertainty and subjective trust, on the attribution of facts, and on *intensional* processes to align remote sources through a small number of localized queries.

Notes and Acknowledgements

Section 7 of this report has been adapted from a workshop article [OAS12] which was written in English, and large parts of this report are intended to be used for a journal presentation of [SAS11]. For these reasons, this report is written in English.

The research has been funded by the European Union’s seventh framework programme, in the setting of the European Research Council grant Webdam, agreement 226513, and the FP7 grant ARCOMEM, agreement 270239.

I wish to thank my supervisor Pierre Senellart for his genuine help, his keen insights, and his impressive efficiency. I am grateful to Fabian Suchanek for a very pleasant and productive stay at MPI and for his constant availability both in person and via email. Thanks to Marilena Oita for our fruitful collaboration and to Mayur Garg for his contribution to the code. Thanks to Serge Abiteboul, David Montoya, and Émilien Antoine for helpful discussion during our “Data Integration at Large” meetings. Thanks to Rainer Gemulla for insight on join functionalities.

Last but not least, thanks to Pierre for his careful proofreading of this report, and to Robin for helpful comments.

1 Background

In this section, I review the general setting of my internship: I present the semantic Web and the problem of ontology alignment.

1.1 Ontologies and the Semantic Web

Motivation. Most of the information on the Web today is written directly for humans in semi-structured HTML documents. The semantic Web is an effort to promote standards for *structured*, semantically-annotated information on the Web, which allows us to trade some of the convenience of natural language text in exchange for the numerous advantages of structured data. Motivations include ensuring interoperability between different organizations, building complex systems which rely transparently on data from various sources, automatically checking the consistency of the data, performing machine reasoning on it to infer knowledge from it, and answering complex queries on it.

RDF. The main standard to state information on the semantic Web is RDF [W3C04d]. An RDF document is a set of *triples*: each triple contains a *subject* position, an *object* position, and a *predicate* position, and each position contains a Uniform Resource Identifier (URI) which identifies a resource (except the object position which may contain a literal value instead, as will be seen later). The triple expresses that the resources occurring at the subject and object positions are related by the resource occurring at the predicate position. In theory, a given resource can occur indifferently at subject, object and predicate positions; however, in practice, resources can be separated in those which appear at the subject and object positions (which we will call *entities*) and those which appear as predicates (which we will call *relations* or *properties*). In our terminology, an *ontology* is simply an RDF document.

URIs. URIs used to identify resources can be of two kinds. They can be the well-known Uniform Resource Locators (URLs), which identify resources by their location indicated as a *scheme*, *domain* (domain name managed by the DNS system), and *path* (managed by the domain owner): for instance, `http://www.gutenberg.org/files/1400/1400-h/1400-h.htm` is the URI for the Project Gutenberg online edition of Charles Dickens' *Great Expectations*. They can also be Uniform Resource Names (URNs) which identify uniquely a resource by a *namespace* and a *namespace-specific string*: for instance, `urn:isbn:0486415864` identifies a certain paper edition of *Great Expectations*. URIs in RDF can be abbreviated with the use of *namespace prefixes*, defining for instance "`dbp:`" to stand for "`http://dbpedia.org/`". In the rest of this documents, no URNs will appear: the frequent use of namespace prefixes to shorten URIs should not be mistaken for URNs.

RDF by example. As an example, consider the following RDF triple (expressed in N-Triples notation, details follow) from the DBpedia [BLK+09] ontology. This triple would be rendered in natural language as "Paris is located in France", where "Paris", "located" and "France" refer to resources managed by `http://dbpedia.org/`.

```
<dbp:resource/Paris> <dbp:ontology/country> <dbp:resource/France> .
```

Our next example indicates that the website of Paris is `http://www.paris.fr/`. Note that the URIs at the predicate and object positions do not point to the DBpedia domain: this illustrates how the semantic Web makes it possible to express assertions between resources managed by independent entities.

```
<dbp:resource/Paris> <http://xmlns.com/foaf/0.1/homepage> <http://www.paris.fr/> .
```

As a final example, consider the following triple, which expresses that "Paris" (the DBpedia resource) has the name "Paris" (the string), with the "`@en`" language indicator to indicate that this is in the English name (which is coincidentally the same as the French name). Other such annotations exist to indicate literal datatypes, but we will not go into the details of this and will remove them from further examples for simplicity. This example illustrates the fact that the object of an RDF predicate can also be a *literal value*.

```
<dbp:resource/Paris> <http://xmlns.com/foaf/0.1/name> "Paris"@en .
```

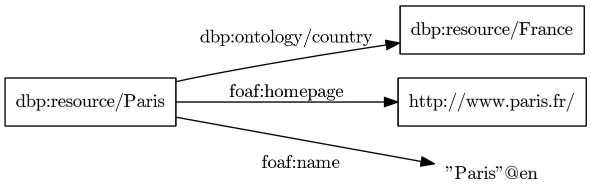

Figure 1: RDF facts from DBpedia about Paris, represented as a graph.

Storage. RDF can be written in different *serialization formats*. The first option is to write it as triples like in the examples above, using the N-Triples notation or supersets of this notation such as Turtle or Notation 3. The second option is to write it as XML. We will not give further details about syntax here: the interested reader is referred to [W3C04a, W3C12, W3C11, W3C04c]. RDF data can be stored directly in a text file in one of the above formats and read sequentially, or it can be accessed by querying specialized databases (*triple stores*) using the SPARQL language.

RDF sources. Semantic Web data can come from several different sources. It can be manually built by humans: either by writing RDF directly, or by annotating unstructured resources with semantic information using RDFa or microformats. It can be produced automatically from existing databases by exporting them to RDF. It can also be produced automatically from unstructured sources using *information extraction* techniques. For instance, the DBpedia [BLK+09] and Yago [SKW07] ontologies are automatically extracted from Wikipedia.

Representations. Beyond the different ways of serializing RDF, there are different possible ways to look at RDF data. First, we can see it as a graph with entities and literals as vertices, relations as edge labels, and one labeled edge per triple. Figure 1 sums up the triples given as examples above. Second, we can see it as a classical database. In this case, we can either think of it as a database with one ternary relation between subjects, predicates, and objects, or we can think of it as a database where each predicate is a binary relation between its subjects and objects.

Schema. So far, we have seen no way to describe the *structure* (or *schema*) of an RDF document. Interestingly, in contrast with traditional relational databases which need to know the structure of the data to figure out how to store it, RDF does not require the user to specify a description of the schema: it can be left implicit and guessed from the data. On the other hand, if the user wishes to provide a schema, the RDF, RDFS [W3C04b] (RDF Schema) and OWL [W3C09] (Web Ontology Language) standards define RDF properties to do this. In a more general way, these predicates are a very expressive language to express *logical constraints* on the data: the expressiveness of the full OWL standard is actually sufficient to make it undecidable, though decidable subsets of OWL exist. Here are some important properties which pertain to the schema of RDF documents:

`rdf:type` This property indicates that a resource is an *instance* of a certain *type*, or *class* in the sense of object-oriented programming. Of course, classes in RDF are also resources. Some standard classes (the class of literal values, the class of all resources) are defined by RDF and RDFS. Entities can have multiple classes.

`rdfs:subClassOf` This property indicates that a class is a subclass of another class. Classes can be subclasses of multiple classes. The meaning of this property is the following: if A is a subclass of B, then all resources which are instances of A are also instances of B. Besides, the property is transitive: if A is a subclass of B and B is a subclass of C, then A is a subclass of C. Note that facts implied by properties such as `rdf:type` and `rdfs:subClassOf` do not need to be materialized in the document or in the schema.
Like for the entity/relation distinction, there is no strong requirement for classes and instances to be disjoint sets, but this assumption usually holds in practice, so we will call *classes* those resources which appear as the object of `rdf:type` or `rdfs:subClassOf`, and we will amend our definition of entities to exclude classes.

`rdfs:subPropertyOf` This property indicates that a property is implied by another property. In other words, if p is a subproperty of p', it means that whenever (x, p, y) holds, then (x, p', y) also holds. Unlike the two previous ones, this property does not occur especially frequently “in the wild”: however, we define it because this notion of subproperty will occur again later.

`owl:sameAs` This property indicates that two different resources have actually the same identity. While such an assertion might seem paradoxical, the use of various distinct URIs to refer to the same thing is widespread. Once again, the presentation of this property is motivated by the fact that the equality of two resources will be a useful notion later.

1.2 Ontology Alignment

In an ideal world, each resource would have exactly one URI, and all ontologies would do their best to reuse existing URIs as much as possible. Obviously, this is not a reasonable expectation. A more realistic view of the semantic Web is the one advocated by the *Linked Data* [Lin] project: a mapping of existing data sources to RDF, and links between these various data sources. This vision is already thrilling, because it would make it possible to integrate independent sources with entirely different schemas and to regroup them in a federated Giant Global Graph (GGG).

However, to achieve this vision, we must have a way to determine links between the data sources, i.e., to *align* them. More specifically, we will say that an *alignment* (or *matching*) between two ontologies can be expressed in RDF as a set of `rdfs:subClassOf`, `rdfs:subPropertyOf`, and `owl:sameAs` statements between resources of the two ontologies. The restriction of the alignment to each of these types of statements is respectively the *class alignment*, the *relation alignment* and the *entity alignment*. Of course, because the ontologies are numerous and large (up to millions of entities), we want to find a way to derive such alignments automatically.

The problem that ontology alignment faces is that two ontologies will use different names (URIs) to talk about the same things. Consider, for instance, the following statements from the DBpedia and Yago ontologies (using the “y:” namespace prefix to stand for the Yago namespace):

```
<dbp:resource/Titanic_(film)> <dbp:ontology/producer> <dbp:resource/Charles_Brackett> .
<y:Charles_Brackett> <y:produced> <y:Titanic_(film)> .
```

A human would quickly identify that those two statements express the same thing, namely, that the movie *Titanic* was produced by Charles Brackett. To do so, we rely on the similarity of URIs; however, strictly speaking, there is no guarantee that `dbp:Charles_Brackett` and `y:Charles_Brackett` have anything to do with each other, or that the way URIs are written have anything to do with the name of the resource which they identify. For instance, in the Internet Movie Database IMDb [Dat] which was adapted as an ontology for Paris ([SAS11]), the same fact in expressed as follows:

```
<imdb:p138992> <imdb:producerOf> <imdb:tt0046435> .
```

While looking at URIs seemed like a good heuristic, in the general case we actually need to look at the facts which give the name of the various resources as literals:

```
<y:Charles_Brackett> <y:hasPreferredName> "Charles Brackett" .
<y:Charles_Brackett> <y:produced> <y:Titanic_(film)> .
<y:Titanic_(film)> <y:hasPreferredName> "Titanic" .

<imdb:p138992> <imdb:label> "Charles Brackett" .
<imdb:p138992> <imdb:producerOf> <imdb:tt0046435> .
<imdb:tt0046435> <imdb:label> "Titanic" .
```

This is easier said than done, however. How do we find out the relation between `y:produced` and `imdb:producerOf`, and between `y:hasPreferredName` and `imdb:label`? Alternatively, how do we identify that the co-occurrence of the two literals “Charles Brackett” and “Titanic” is really an interesting basis to align resources?

Another issue is that the names may not match exactly. For instance, compare the name given to Charles Brackett in Yago (above) and in DBpedia (below):

```
<dbp:resource/Charles_Brackett> <foaf:name> "Charles William Brackett"@en .
```

Yet another issue is that the structure of the two ontologies might not match fact for fact as they did in the simple case above. As a more complicated example, here is the fact indicating Douglas Adams' birth date in DBpedia and the three facts which indicate it in the British Library ontology [Lib]. Notice how DBpedia gives the exact date while the British Library only gives the year, and how the British Library has an intermediate entity to represent the birth of Douglas Adams.

```
<dbp:resource/Douglas_Adams> <dbp:ontology/birthDate> "1952-03-11".
<bnb:person/AdamsDouglas1952-2001> <bio:event> <bnb:person/AdamsDouglas1952-2001/birth> .
<bnb:person/AdamsDouglas1952-2001/birth> <rdf:type> <bio:Birth> .
<bnb:person/AdamsDouglas1952-2001/birth> <bio:date> "1952".
```

These examples illustrate why ontology matching can be a challenging task.

2 Paris

In this section, we present the Paris (Probabilistic Alignment of Relations, Instances, and Schema) system for ontology alignment, which is described in more detail in [SAS11]. The Paris system was developed by Fabian Suchanek, my supervisor Pierre Senellart, and Serge Abiteboul.

2.1 Informal Presentation

The Paris system aligns two ontologies O_1 and O_2 in the sense of the previous section, and does so without prior knowledge on the ontologies, without training data, and without parameter tuning. It assumes, however, that there are no duplicate entities within the ontologies that it aligns. It does not produce crisp matchings, but gives a confidence score to matchings, following a probabilistic model.

The general working of Paris is as follows: we keep alignment scores $\Pr(x \equiv y) \in [0,1]$ for every couple of entities or literals $(x, y) \in O_1 \times O_2$, and alignment scores $\Pr(r \subseteq r') \in [0,1]$ for every couple of relations (r, r') from each ontology (formally, $(r, r') \in O_1 \times O_2 \cup O_2 \times O_1$). We start with an initial matching which is set to zero for pairs of entities, to a literal equality function for pairs of literals, and to a small constant for pairs of relations. From this, we apply formulae to get a new matching, and iterate this process until we reach a fixpoint, which is the result of the computation. We then compute the class alignment from the result. In theory, the class alignment could cross-fertilize the entity alignment, but this does not work well in practice.

The formulae to update the alignments are designed to make the relation matching and entity matching *cross-fertilize*, i.e., each one draws on the other. Their informal wording is as follows:

Entities. Two entities are matched if they are linked to a matched entity pair by a matched pair of relations. This is refined by a notion of *relation functionality* which is computed from the data. See Figure 2.

Relations. Two relations are matched if whenever one occurs between two entities then the other one occurs between the matching entities. See Figure 3.

Functionalities. A relation r is said to be *functional* if it behaves as a function from its first argument to its second argument, i.e., if for all x there exists at most one y such that $r(x, y)$ holds. *Inverse functionality* is defined symmetrically with the second and first arguments.

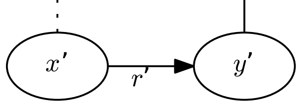

Figure 2: Entity alignment rule. Fact $r(x, y)$ holds in O_1 and fact $r'(x', y')$ holds in O_2. The solid edge represent an existing alignment, the dotted edge a new alignment to do if the relations r and r' are aligned and inverse functional.

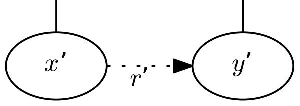

Figure 3: Relation alignment rule. Fact $r(x, y)$ holds in O_1 The solid edges represent existing alignments, the dotted edge is checked as evidence that the alignment $r \subseteq r'$ should be made or not.

Functionality is an important notion to indicate that a relation expresses a unique attribute of an entity, e.g., the country of birth of a person, the owner of a phone number, the capital of a country, etc. The functionality of a relation can be specified as part of the ontology schema using OWL predicates: this is analogous to *functional dependencies* in relational databases. It is a helpful tool for ontology alignment: if two entities y and y', and two relations r and r', are aligned, if r and r' are inverse functional, and if it holds that $r(x, y)$ and $r'(x', y')$ for some x, x', then x and x' have to be aligned, because the inverse functionality of r and r' ensures that they must represent the same object.

However, we do not wish to assume that the ontologies which we will align with Paris provide a description of their schema which indicates functionality constraints. Besides, relations which are *almost* (but not exactly) functional, such as the date of birth or even the nationality, can still provide useful information about the alignments to perform: for the example of the previous paragraph, the more inverse functional r and r' are, the more x and x' should be aligned. To achieve this, Paris defines a "fuzzy" functionality for relations (between 0 and 1) which is estimated from the data by looking at each relation's occurrences.

2.2 Related Work

Most existing work on ontology alignment focuses only on schema alignment (relations and classes) or on instance matching (entities). Tools which only perform schema alignment such as [ADMR05] work very differently from Paris because they use a schema description with rich constraints rather than data to guide the alignment: they rely on structural similarity of schemas and string similarity between relation and class names.

As for systems which only perform entity alignment, Paris shares some techniques with [HPUZ10] which aligns based on predicates which were identified as functional from the data (but does not propagate alignments like Paris does).

There are very few *holistic* systems which perform schema and entity matching simultaneously. Existing systems such as RiMOM [LTLL09] or Illiads [UGM07] either have not been tested on large ontologies or are focused on domain-specific strategies rather than generic approaches. Others such as Micu [Zai10] are not really holistic but built by combining existing entity matching and schema matching techniques. Another very recent system is SiGMa [LJPD+12] which is able to achieve impressive running times through greedy choices but requires a manual seed alignment of relations.

2.3 Formal Description

Preliminary notations. We assume that a similarity function L is defined on pairs of literals such that $L(l, l') \in [0, 1]$ denotes an equality score for l and l'. Of course, $\forall l,\ L(l, l) = 1$. For convenience, we will always assume that whenever a relation r occurs in an ontology O, then the relation r^- is also implicitly defined by O, where $r^-(x, y)$ holds iff $r(y, x)$ holds.

Alignments. We define the entity and relation alignments at step n to be $E^n = (\Pr^n(x \equiv x'))_{(x,x') \in O_1 \times O_2}$ and $R^n = (\Pr^n(r \subseteq r'))_{(r,r') \in O_1 \times O_2 \cup O_2 \times O_1}$, where x, x' denote entities or literals and r, r' denote relations. For all n, for all literals $(l, l') \in O_1 \times O_2$ and entities $(x, x') \in O_1 \times O_2$, we will have $\Pr^n(l \equiv l') = L(l, l')$ and $\Pr^n(l \equiv x') = \Pr^n(x \equiv l') = 0$. In other words, only alignments between proper entities need to be stored, alignments involving literals are either fixed to 0 or determined from L.

We define the initial alignment E^0, R^0 as $\forall (x, x') \in O_1 \times O_2$, $\Pr^0(x \equiv x') = 0$ and $\forall (r, r') \in O_1 \times O_2 \cup O_2 \times O_1$, $\Pr^0(r \subseteq r') = \theta$ for some small constant θ.

Functionalities. Given a relation r in an ontology O and an entity x in O which appears as first argument for r (i.e., $\exists y,\ r(x, y)$), we define the *local functionality* of r at x as:

$$fun(r, x) = \frac{1}{\left|\{y \in O \mid r(x, y)\}\right|}$$

The functionality of r is defined as the harmonic mean of its local functionalities over all entities $x \in O$ where they are defined, which turns out to have the following closed form:

$$fun(r) = \frac{\left|\{x \in O \mid \exists y \in O,\ r(x, y)\}\right|}{\left|\{(x, y) \in O^2 \mid r(x, y)\}\right|}$$

We define the inverse functionality fun^{-1} of a relation r as the functionality of r^-.

Equations. Now, we write the equations which define the entity and relation alignments at step $n+1$ from those at step n. See [SAS11] for alternative design choices for these equations, including some which take negative evidence into account.

$$\begin{aligned}\Pr^{n+1}(x \equiv x') = 1 - \prod_{\substack{r(x,y)\\ r'(x',y')}} &\left(1 - \Pr^n(r' \subseteq r) \times \mathit{fun}^{-1}(r) \times \Pr^n(y \equiv y')\right)\\ &\times \left(1 - \Pr^n(r \subseteq r') \times \mathit{fun}^{-1}(r') \times \Pr^n(y \equiv y')\right)\end{aligned} \tag{1}$$

$$\Pr^{n+1}(r \subseteq r') = \frac{\sum_{r(x,y)} \left(1 - \prod_{r'(x',y')} \left(1 - (\Pr^n(x \equiv x') \times \Pr^n(y \equiv y'))\right)\right)}{\sum_{r(x,y)} \left(1 - \prod_{x',y'} \left(1 - \Pr^n(x \equiv x') \times \Pr^n(y \equiv y')\right)\right)} \tag{2}$$

See [SAS11] for a probabilistic interpretation which justifies these equations. We give further insight into this in Appendix C, including alternative design choices for the equations guided by this interpretation.

Result. The final alignment E^∞, R^∞ is the fixpoint of the two above equations when starting at E^0, R^0. Note that we have not given any theoretical guarantees for the existence of this fixpoint; this will be discussed in Section 5. Class alignments are computed from E^∞, R^∞; we omit the formal computation of class alignments as we will not use it in the rest of this report, the reader is referred to [SAS11] for details about this step.

2.4 Implementation

The implementation of PARIS is a straightforward mapping of the equations to code. In this section, I describe the original implementation of PARIS, without my changes that will be introduced in future sections[1].

To ensure that the appropriate facts and entities can be looked up efficiently, the ontologies are kept in a Berkeley database with adequate indexes. Because of the numerous random accesses, this database has to reside on a SSD to ensure acceptable run times.

Algorithm 1: Entity alignment computation

```
   Data: $O_1$, $O_2$, $E^n$, and $R^n$
   Result: $E^{n+1}$
 1 for $(x, x') \in O_1 \times O_2$ do
 2     $Q[x, x'] := 1$;                                  /* Stores the product for Equation (1) */
 3 for $x \in O_1$ do
 4     for $r, y$ such that $r(x, y)$ do
 5         for $r' \in O_2$ do
 6             for $y'$ such that $E^n[y, y'] > 0$ do
 7                 for $x'$ such that $r'(x', y')$ do
 8                     $Q[x, x'] := Q[x, x'] \times (1 - fun^{-1}(r) \times E^n[y, y'] \times R^n[r', r])$;
 9                     $Q[x, x'] := Q[x, x'] \times (1 - fun^{-1}(r') \times E^n[y, y'] \times R^n[r, r'])$;
10 for $(x, x') \in O_1 \times O_2$ do
11     $E^{n+1}[x, x'] := 1 - Q[x, x']$;
```

The entity alignment computation is given as Algorithm 1. The main difference between the PARIS model and the PARIS implementation is the following: for every entity, the implementation only keeps one equality candidate at each step, chosen at random from those which achieve the best score. Formally, as soon as E^n is computed, it is replaced by a variant E'^n where for every x, there is at most one y_0 such that $\Pr'^n(x \equiv y_0) > 0$, in which case y_0 should realize $\max_y P^n(x \equiv y)$ and $\Pr'^n(x \equiv y_0) = \Pr^n(x \equiv y_0)$; for every y, the symmetric constraint is imposed. This is done to reduce memory usage and to make the computation faster at the next

[1]However, I do not take into account a bug present in the original version of PARIS where quantity d in Algorithm 2 was incorrectly computed.

steps (because only one equality candidate will need to be considered). For this reason, the **for** loop starting at line 6 in Algorithm 1 only iterates on 0 or 1 element in practice.

Another efficiency tweak is the fact that relation alignment scores, entity alignment scores, and inverse functionalities will be considered as equal to zero if they are less than some small constant θ. Relations which have too low a functionality will not be considered. Yet another trick used to improve the runtime is to store, for each r, a list of the candidate r' to try out. Initially, all r' are candidates, but a candidate gets discarded when it does not result in enough useful alignments (i.e., we stop considering r' relations for r whenever this choice of r' does not make lines 8 and 9 run a sufficient number of times).

Algorithm 2: Relation alignment computation for alignments in $O_1 \times O_2$

```
Data: $O_1$, $O_2$, and $E^{n+1}$
Result: $R^{n+1}$ in one direction
1  for $r \in O_1$ do
2      for $r' \in O_2$ do
3          $N[r'] := 0$;                     /* Stores the numerator for equation (2) */
4          $R^{n+1}[r, r'] := 0$;
5      $d := 0$;                             /* Stores the denominator for equation (2) */
6      for $x, y$ such that $r(x, y)$ do
7          $v := 1$;
8          for $r' \in O_2$ do
9              $T[r'] := 1$;
10         for $x', y'$ such that $E^n[x, x'] > 0$ and $E^n[y, y'] > 0$ do
11             $v := v \times (1 - E^n[x, x'] \times E^n[y, y'])$;
12             for $r'$ such that $r'(x', y')$ do
13                 $T[r'] := T[r'] \times (1 - E^n[x, x'] \times E^n[y, y'])$;
14         for $r' \in O_2$ do
15             $N[r'] := N[r'] + (1 - T[r'])$;
16         $d := d + (1 - v)$;
17     if $n > 0$ then
18         for $r' \in O_2$ do
19             $R^{n+1}[r, r'] := N[r']/d$;
```

The relation alignment computation is given as Algorithm 2. Unlike the entity alignment, it is performed in both directions, i.e., the algorithm is called for alignments in $O_1 \times O_2$ and then for alignments in $O_2 \times O_1$. Once again, in practice, entity alignment scores are considered to be zero if they are less than θ.

The final alignment is determined by iterating these algorithms until the alignment does not change anymore or until a maximal number of iterations is reached.

2.5 Results and Evaluation

Paris is experimentally evaluated in [SAS11] on three main tasks: aligning benchmark datasets from the Ontology Alignment Evaluation Initiative, 2010 [EFM+10]; aligning Yago and DBpedia; and aligning Yago and IMDb.

Performance is measured by comparing the alignment produced by Paris to a *gold standard* which is assumed to represent the truth for this matching task. In some cases, the gold standard alignments were provided with the datasets (OAEI), produced manually (IMDb–Yago relations), derived from the URIs (Yago–DBpedia entities), or computed through ad-hoc means (IMDb–Yago entities). In other situations, no gold standard was available so an incomplete gold standard was devised by evaluating the results of Paris by hand, possibly by sampling them if they were too large (Yago–DBpedia and IMDb–Yago class alignments).

The quality of the Paris matching with respect to the gold standard can be evaluated in terms of *precision*, in terms of *recall*, or in terms of *F-measure*. Precision measures the proportion of produced matches which are correct, recall measures the proportion of correct matches which were produced, and F-measure is the harmonic mean of those two scores. Formally, we see the gold standard as a set G of correct matchings to produce and the results R as a set of matchings effectively produced, and we define:

$$\mathit{precision} = \frac{|R \cap G|}{|R|} \qquad \mathit{recall} = \frac{|R \cap G|}{|G|} \qquad F = \frac{2 \times \mathit{precision} \times \mathit{recall}}{\mathit{precision} + \mathit{recall}}$$

The results of Paris on the various tasks and the various datasets are summarized in Table 1 in Appendix B.

3 Performance Improvements

Paris has been criticized for the time it takes to run [LJPD$^+$12]. A contribution of my internship is to improve the efficiency of the Paris implementation through a variety of techniques. This section does not discuss my more radical changes to Algorithms 1 and 2 which were motivated by the need to add support for join relations but were also helpful to improve performance: they will be presented in Section 4.3.

In-Memory Storage. The first optimization to do was to move the ontologies from the SSD to main memory in order to ensure that Paris was no-longer IO-bound but CPU-bound. To do so, I replaced the Berkeley DB store by a custom in-memory store which manages directly the various indexes required to perform the queries of Algorithms 1 and 2 efficiently. Another possible approach would have been to make use of an existing in-memory triple store such as RDF3X [NW08]. The use of a custom in-memory store has the advantage of ensuring that the memory overhead is minimal (because we can control precisely how the data is stored).

To make it possible to run the alignment tasks, my supervisor and I studied the possible hardware choices and settled on the X79A-GD45 motherboard by MSI which supports 8 DDR3 slots; however, this forced us to use Intel's recent LGA2011 socket, for which few affordable CPUs were available (we chose the Intel Core i7-3820). We assembled the machine ourselves.

Threading. The main bottleneck in the Paris computation after moving to an in-memory store is the main loop of Algorithm 1. An important observation to make is that this loop can be parallelized easily: taking advantage of this would be useful on multi-cores architectures; besides, it leaves open the possibility of distributing the computation between different machines.

I implemented the loop on entities of O_1 as a multithreaded computation, using a thread pool to avoid the cost of spawning new threads for each entity. No care needs to be taken when performing parallel accesses to O_1 and O_2 since only reads are performed: the only operations to synchronize are the reads and writes in $Q[x, x']$, which is easy because the various threads use different, fixed values for x.

Changing Data Structures. Changing the data structures in several places lead to efficiency gains. For instance, the original Paris stores the relation alignments as triples of two relations and a floating point score which are kept in a Berkeley database and indexed by sub- and super-relation. The natural translation of this to make it run in-memory is to use `MultiMaps` as indexes from sub- and super-relations to the equality pairs, but a much more efficient choice is to use a native array of dimension two. (The relation alignment matrix is sufficiently small and dense to make this preferable to a sparse solution involving a hash table.)

Results. To present the improvement in run times between the original implementation and the new implementation, I compare them on the DBpedia–Yago alignment task which is one of the main experiments presented in [SAS11]. The times reported for the original version are those indicated in the paper, on an older (and smaller) version of DBpedia and Yago. Because the datasets have changed, the results of the two versions are not identical, but they have similar F-measure. Note that because of the high RAM requirement, the hardware used is not the same either: the results are therefore not directly comparable, but they give a idea of the general efficiency gain. The results are presented in Table 2 in Appendix B.

4 Join Relations

One extension to Paris that I developed during my internship is the support for join relations. In this section, I define join relations and present related work on this topic, before presenting the changes that must be made to Paris to support them. I conclude with a presentation of the practical difficulties posed by join relations.

4.1 General Presentation

As explained in Section 1.2, ontology alignment can be hard to perform because the two ontologies do not have the same structure.

Join relations are a natural idea to make it possible to align ontologies with a different structure. Given a sequence of relations $r_1, ..., r_n$ (not necessarily distinct) in an ontology O, we define the join relation $(r_1, ..., r_n)$ as follows:

$$\forall x, y \in O,\ (r_1, ..., r_n)(x, y) \Leftrightarrow \exists z_1, ..., z_{n-1},\ r_1(x, z_1) \wedge r_2(z_1, z_2) \wedge \cdots \wedge r_n(z_{n-1}, y) \tag{3}$$

Note that the number of possible choices for the existentially quantified variables does not matter. The name of “join relations” comes from the fact that if you consider $(r_1, ..., r_n)$ as a binary relation in the relational algebra, then its table is what you obtain by considering the tables of the binary relations representing r_1, ..., r_n, renaming the columns so that the second column of each r_i matches the first column of r_{i+1}, joining the tables, and projecting on the first column of r_1 and on the second column of r_n.

We have presented joins in logical terms and in relational terms, but the graph interpretation may be the most intuitive: the facts for a join relation $(r_1, ..., r_n)$ are simply the endpoints of the paths labeled by $(r_1, ..., r_n)$ in the graph.

4.2 Related Work

The alignment of join relations that we want to compute with Paris could be seen as a simplified case of *schema mapping* if we see the ontologies as relational databases. A schema mapping [Kol05] is a set of logical constraints which describes the schema of a database as a function of the schema of another database. In this context, the problem of learning schema mappings from examples (given two database instances, find a schema mapping from one to the other which is consistent with the data) has already been studied: see for instance [tCDK12] for a theoretical study of this learning problem. Our setting is different, however, because we are interested in imperfect mappings (i.e., we are willing to tolerate some errors), and because the alignment of entities (i.e., equality on the database instances) is not crisp. For a study of the complexity of inferring the optimal schema mapping from examples, where the optimality is defined by the minimal costs in terms of a combination of the mapping size and the number of errors to correct, see [GS10]. This work does not lead to any practical solution.

The more general problem of understanding the structure of a set of examples to produce a logic program which generates those examples is known as *Inductive Logic Programming* [LD94]: however, in this setting, we need negative evidence to guide the learning, which is not directly available in our case. Many problems from data mining, for instance association rule learning, can be related to Inductive Logic Programming. Such techniques can be applied to knowledge bases, see for instance [LMC11] or the ongoing work of Fabian Suchanek’s PhD student Luis Galárraga.

4.3 Modifications to Algorithms 1 and 2

We will now describe how the support for join relations can be added to Paris.

The main change when join relations are supported is that it adds a large number of relations which, because of memory constraints, are not *materialized*, i.e., we cannot compute them and add them to the in-memory representation. For this reason, we cannot request all facts for a given join relations without an unreasonable amount of computation. This observation means that the entity and relation alignment computations need to be changed. Algorithm 1 can be made to work simply by changing the order of the loops and tweaking them slightly. However, Algorithm 2 cannot be adapted so easily.

The solution that I propose is to perform the entity and relation alignment simultaneously. The crucial observation is that, as we iterate over fact pairs to perform entity alignment, we are seeing almost all of the information that we need in order to perform relation alignment: while we compute E^{n+1}, we can review the evidence in E^n to compute R^{n+1}. This means that R^{n+1} will depend on E^n instead of E^{n+1}. See Algorithm 3 for the combined entity and relation alignment algorithm. When we iterate on all facts with a fixed entity as first or second argument in lines 6 and 12, this should be taken to mean “all facts with a simple relation r or an interesting join relation r” for some measure of interestingness.

Algorithm 3: Entity and relation alignment computation

```
Data: $O_1$, $O_2$, $E^n$, and $R^n$
Result: $E^{n+1}$ and $R^{n+1}$
for $(x, x') \in O_1 \times O_2$ do
    $Q[x, x'] := 1$;                                   /* Stores the product for equation (1) */
$N :=$ new HashMap();                                  /* Stores the numerator for equation (2) */
$D :=$ new HashMap();                                  /* Stores the denominator for equation (2) */
for $x \in O_1$ do
    for $r, y$ such that $r(x, y)$ do
        $v := 1$;
        $T :=$ new HashMap();
        for $y'$ such that $E^n[y, y'] > 0$ do
            for $x'$ such that $E^n[x, x'] > 0$ do
                $v := v \times (1 - E^n[x, x'] \times E^n[y, y'])$;
            for $r', x'$ such that $r'(x', y')$ do
                $Q[x, x'] := Q[x, x'] \times (1 - fun^{-1}(r) \times E^n[y, y'] \times R^n[r', r])$;      /* Direction 1 */
                $Q[x, x'] := Q[x, x'] \times (1 - fun^{-1}(r') \times E^n[y, y'] \times R^n[r, r'])$;     /* Direction 2 */
                if $T[r']$ was not initialized then
                    $T[r'] := 1$;
                $T[r'] := T[r'] \times (1 - E^n[x, x'] \times E^n[y, y'])$;      /* Align r and r' */
        for $r'$ such that $T[r']$ was initialized do
            if $N[r, r']$ was not initialized then
                $N[r, r'] := 0$;
            $N[r, r'] := N[r, r'] + (1 - T[r'])$;          /* Update numerator for r(x, y) */
        if $D[r]$ was not initialized then
            $D[r] := 0$;
        $D[r] := D[r] + v$;                                /* Update denominator for r(x, y) */
for $(x, x') \in O_1 \times O_2$ do
    $E^{n+1}[x, x'] := 1 - Q[x, x']$;
for $(r, r') \in O_1 \times O_2$ do
    $R^{n+1}[r, r'] := N[r, r'] / D[r]$;
```

The notion of "interestingness" to use when exploring joins is an important question. In our implementation, we used a simple depth-first traversal of the graph at each node, bounded by a maximum depth. Note that this traversal must store the various facts that it encounters and discard duplicates, because a same fact can correspond to different paths in the graph since we do not care about intermediate entities (see Figure 4 in Appendix B for an example of this).

To parallelize the computation of Algorithm 3, we can share Q across the various threads as before, but we cannot share the numerators and denominators N and D across the threads because of possible race conditions. Instead, we make each thread compute its own N and D from the entities that it took care of, and we sum the N and D once all threads have finished following Equation (2). This can be thought of as a simple MapReduce operation: the Map step maps entities to values of N and D accounting for the facts involving this involving this entity as first argument, and the Reduce step sums the D and N.

One advantage of Algorithm 3 over Algorithms 1 and 2 is that the only accesses made to O_2 are of the form "get all facts involving a certain literal or entity" or "find all literals similar to a certain literal". We do not need to iterate over all relations, or find all facts for a given relation. This could be useful to apply Paris to an *intensional* setting, where one of the ontologies to align is stored on a distant server and can only be accessed through simple requests: Algorithm 3 can proceed by querying the remote ontology for facts about matched entities, and discover the relations of the remote ontology and align them on the fly. This is also useful when one ontology is much smaller than the other, for instance in the case of Section 7: we will only explore the small part of the big ontology which is close to literals which match in the small ontology, which ensures good running time.

One drawback of Algorithm 3, however, is that the relation alignment is only computed in one direction (i.e., determining that a relation of O_1 is included in a relation of O_2), like with Algorithm 2, so we need to run this algorithm a second time (switching the roles of O_1 and O_2) to obtain a relation alignment in the second direction. It seems that the relation alignment rule makes this hard to avoid, for the following reason: whenever x and y in O_1 align with x' and y' in O_2 and there is a fact $r'(x', y')$ but no fact $r(x, y)$, then we must account for $r'(x', y')$ in the denominator of Equation (2) for r'; however, when exploring O_1, we have no reason to see this fact because x and y are not linked by any fact. An interesting question that we leave open is to know whether it is possible to compute an approximation of the relation alignment in both directions by performing Algorithm 3 only once.

4.4 Functionality Estimation

To perform the computation, we need to know the functionality of join relations. Determining them is an interesting subproblem that I did not manage to solve satisfactorily. We can count the number of occurrences and the size of the domain of joins like we do for simple relations, but this is already a challenging task on large ontologies such as YAGO (remember that join relations are not materialized). It seems that the functionality of join relations should be related to the functionality of the individual relations which comprise them, but a closer look suggests that this probably not the case. For a join of two relations (r_1, r_2), the functionality can be very different depending on how the join occurs, i.e., depending on the intersection of r_1's codomain and r_2's domain. See Figure 5 in Appendix B if you need an illustration. In the light of such examples, it seems unlikely that useful, tight bounds can be derived.

For this reason, as a simple heuristic, we set the functionality of a join relation to be the minimum of the functionalities of the individual relations being joined.

It is interesting to note that the related problem of estimating the *cardinality* of joins (i.e., the number of facts for a join relation) has been studied in the context of triple stores because it can be used to estimate the size of intermediate results to help choose query execution plans. See for instance [NW08] or more specifically [NM11].

4.5 Practical Issues and Further Work

To select interesting joins to perform, we tried to limit first to all joins with a length of 2. In practice, we did not manage to run an alignment task with this setting on large ontologies, and only ran it in a setting where one of the ontologies was small (see Section 7).

To actually perform the alignment of large ontologies using joins, a more clever approach would be needed to select which joins are promising and which joins should not be explored, and possibly with which relations we should try to align a certain join relation. Algorithm 3 allows us to estimate on the fly the support of join relations and the current confidence of a relation alignment. This could be used to stop considering alignments early if they do not seem promising.

Another issue with join relations is that we should not pay attention to relations with insufficient support. However, it is not easy to know how the support should be bounded, especially if we wish to retain the parameter-free nature of PARIS. One option, inspired by minimum description length and by [GS10], would be to say that an alignment should be kept only if the extent to which it explains the data (i.e., the amount of data that it summarizes) is larger that the space it takes to describe this alignment.

Despite those shortcomings, the addition of joins make it possible for PARIS to align the simple cases of joins which occur in the restaurant dataset from the Ontology Alignment Evaluation Initiative, 2010 [EFM+10], thus bringing the recall of relation alignments on this dataset from 66% to 100% while maintaining 100% precision.

5 Convergence Analysis

There is no guarantee of convergence for PARIS, only experimental proof. This section outlines possible ideas to prove the convergence in simplified models of PARIS.

As a general remark, let us first notice that if we fix the relation alignment score, then the entity alignment scores converge, because they are bounded (in $[0, 1]$) and increase component-wise. Formally, since we start to iterate at the zero vector, then it must be the case that for all $(x, x') \in O_1 \times O_2$, $\mathrm{Pr}^1(x \equiv x') \geq P^0(x \equiv x') = 0$, and the entity alignment equation guarantees that if $P^{i+1}(x \equiv x') \geq P^i(x \equiv x')$ then $P^{i+2}(x \equiv x') \geq P^{i+1}(x \equiv x')$, so a trivial induction proves our claim. However, this does not give us any intuition about the meaning of the fixpoint.

In this section, we present a simplification of the Paris entity alignment equation (Equation (1)) and try to link it to a variant of PageRank. We then present an alternate entity alignment equation suggested by this analogy. Conversely, a simple example on non-convergence in a specific case is given in Appendix D.

5.1 A Simpler Model

In this subsection, we fix the relation alignment scores and we consider a simplified model:

- Couples of relations are either aligned (with score 1) or not aligned (with score 0).
- Relations are either inverse functional (with score 1) or not inverse functional (with score 0).
- We propagate alignments in a one-directional fashion, i.e., we do not symmetrize, we only take into account $\mathit{fun}^{-1}(r)$ and $r \subseteq r'$.

In this simplified setting, the entity alignment equation rewrites to:

$$\mathrm{Pr}^{n+1}(x \equiv x') = 1 - \prod_{\substack{r(x,y)\\ r'(x',y')\\ \mathit{fun}^{-1}(r)=1\\ r' \subseteq r}} \left(1 - \mathrm{Pr}^n(y \equiv y')\right) \tag{4}$$

We can simplify (4) by looking at the log-probabilities defined as follows and designed so that as $\mathrm{Pr}^n(x \equiv x')$ takes values in $[0, 1]$, $\mathrm{LPr}^n(x \equiv x')$ takes values in $[0, +\infty]$:

$$\mathrm{LPr}^n(x \equiv x') := -\log(1 - \mathrm{Pr}^n(x \equiv x')) \tag{5}$$

Equation (4) rewrites to:

$$\mathrm{LPr}^{n+1}(x \equiv x') = \sum_{\substack{r(x,y)\\ r'(x',y')\\ \mathit{fun}^{-1}(r)=1\\ r' \subseteq r}} \mathrm{LPr}^n(y \equiv y') \tag{6}$$

To simplify this equation further, we will formalize a representation of ontologies as labeled graphs: we write $O_1 = (V_1, R_1, E_1)$ where V_1 are the entities and literals, R_1 are the relations, and $E_1 \subset V_1 \times R_1 \times V_1$ are the facts, and likewise for O_2.

Now, consider the product graph $G = (V_1 \times V_2, R_1 \times R_2, E)$ with:

$$E = \left\{((x, x'), (r, r'), (y, y')) \mid (x, r, y) \in E_1, (x', r', y') \in E_2, \mathit{fun}^{-1}(r) = 1, r' \subseteq r\right\} \tag{7}$$

Except for the conditions on the functionality and on relation alignments, this is exactly the product automata construction if you see O_1 and O_2 as automata on the alphabets R_1 and R_2. Now, let us rewrite Equation (4), writing (x, x') instead of $(x \equiv x')$:

$$\mathrm{LPr}^{n+1}(x, x') = \sum_{((x,x'),(r,r'),(y,y')) \in E} \mathrm{LPr}^n(y, y') \tag{8}$$

Some of the terms of this sum come from neighboring entity alignments, and other come from neighboring literal alignments. Let us distinguish those two cases by defining:

$$L(x, x') = \sum_{\substack{((x,x'),(r,r'),(l,l')) \in E\\ l \text{ and } l' \text{ literals}}} \mathrm{LPr}^n(l, l') \tag{9}$$

Remember that $\mathrm{LPr}^n(l, l')$ is a constant denoting the similarity of l and l', and $\mathrm{LPr}^n(y, l') = \mathrm{LPr}^n(l, y') = 0$ for all entities y, y'. Define $G' = (V_1 \times V_2, R_1 \times R_2, E')$ and $E' = E \backslash \{(x, r, y) \mid x \text{ or } y \text{ literal}\}$. Write M the adjacency matrix of G' (where rows and columns are indexed by couples of proper entities in $O_1 \times O_2$). We will see L and the LPr^n as vectors indexed by such couples. We can therefore write:

$$\mathrm{LPr}^{n+1} = M \,\mathrm{LPr}^n + L \tag{10}$$

Hence, the PARIS computation converges if and only if the LPr series defined in this way converges in $[0, +\infty]$, with $\mathrm{LPr}^0 = \vec{0}$.

The point of reformulating the PARIS computation in this way is to compare it to linear systems such as PageRank. To the difference of PageRank, however, M is not stochastic, so there is no conservation of mass (but diverging to $+\infty$ still translates as convergence for Pr^n), and there is an affine L term which biases the computation towards alignments which are close to couples of similar literals.

More specifically, we can link this model to the use of Green measures in [OS07], but in our case we use L to bias the computation instead of a Dirac measure on only one node.

5.2 A Different Model

It seems natural to lift the restrictions that we made on the simpler model so that M can have arbitrary coefficients in $[0, 1]$ instead of $\{0, 1\}$. Sadly, doing this directly will not work. For concision, we will stay in the unidirectional model, and will write $A(r, r') = \mathrm{Pr}^n(r' \subseteq r) \times \mathit{fun}^{-1}(r)$. The translation of Equation (1) to log probabilities does not give what we intend:

$$\mathrm{LPr}^{n+1}(x \equiv x') = \sum_{\substack{r(x,y)\\ r'(x',y')}} -\log\left(1 - A(r, r')\,\mathrm{Pr}^n(y \equiv y')\right) \tag{11}$$

Our hope was that there would exist a function $B(r, r')$ such that:

$$\mathrm{LPr}^{n+1}(x \equiv x') = \sum_{\substack{r(x,y)\\ r'(x',y')}} B(r, r')\,\mathrm{LPr}^n(y \equiv y') \tag{12}$$

If we want to obtain this, then we should replace Equation (1) by:

$$\mathrm{Pr}^{n+1}(x \equiv x') = 1 - \prod_{\substack{r(x,y)\\ r'(x',y')}} \left(1 - \mathrm{Pr}^n(y \equiv y')\right)^{B(r,r')} \tag{13}$$

Sadly, if we do this, we lose the probabilistic interpretation of Equation (1). The probabilistic interpretation was, informally: "x and x' are aligned if one of the following events (that we assume to be independent) is true: for some $r(x, y)$ and $r'(x', y')$, y and y' are aligned and the alignment should be propagated over (r, r')". In other words, $A(r, r')\,\mathrm{Pr}^n(y \equiv y')$ is to be interpreted as the conjunction of "$r' \subseteq r$ and r is inverse functional" and "y and y' are aligned". By contrast, the alternative equation that we propose does not have a clear probabilistic interpretation.

An interesting question for further work would be to find out the difference in performance between the use of Equation (1) with its probabilistic approach and the use of Equation (13) with the interpretation that we described.

5.3 Understanding the Full PARIS

To prove the convergence of the whole PARIS, it would be necessary to take into account the fact that relation alignment scores vary. However, relation alignment scores are not increasing, but can vary arbitrarily, so the simple argument for convergence given at the beginning of this section does not apply anymore. It seems challenging to study the convergence of the sequence (E^n, R^n); it would perhaps be more promising to fix R^n, compute the limit for entity alignments, compute R^{n+1}, compute the limit for entity alignments, compute R^{n+2}, etc., and prove that this whole process converges. To do so, we would need to understand to what extent the relation alignment computation "respects" the entity alignment and leads to convergence of the whole process.

6 Approximate Literal Matching

In this section, we present an improved way to match literals in Paris through *shingling*. The shingling technique itself was implemented by Mayur Garg (see Section 6.4), I took care of the integration with Paris.

6.1 Motivation

The alignments computed by Paris are initialized ("bootstrapped") by an alignment on literals defined as a similarity function L giving, for any couple of literals x, y, an equality score $L(x, y) \in [0, 1]$. The simplest such function is the exact equality function such that $L(x, x) = 1$ and $L(x, y) = 0$ if $x \neq y$. The original implementation of Paris ([SAS11], sections 5.3 and 6.3) uses the exact equality function with simple normalization techniques: remove language information and datatypes, and (for the datasets where it helps) lowercase and remove punctuation.

The simple approach has limitations, however, because literals in the two ontologies might differ slightly because of spelling variations and typos. For instance, where one ontology may represent the name of Charles Dickens as "Charles Dickens", another one may write "Dickens, Charles", "C. Dickens", or, in the case of ontologies extracted automatically, "by Charles Dickens". The simple approach will consider all these literals to be different, whereas we would want to notice that they are similar and give them a non-zero score to use them in the alignment.

This suggests that a more elaborate literal similarity function is needed. This problem is challenging, because the literals in the ontologies can be of a very diverse nature: they can represent strings, of course, but also numbers, dates, date ranges, etc. We study alternatives designed for the case of strings, because it is a common case and gives a reasonable approximation for other literal types.

6.2 Possible Approaches

There are several well-known string similarity measures:

Levenshtein similarity The Levenshtein distance computes the minimal number of insertions, substitutions and deletions to go from one string to the other. It is computed with a simple dynamic programming algorithm which runs in time proportional to the product of the length of the two strings. The Levenshtein distance can be normalized to $[0, 1]$ by dividing it by the length of the longest of the two strings, and can be turned in a similarity measure by taking the similarity to be one minus the normalized distance.

Damerau–Levenshtein similarity The Damerau–Levenshtein distance is defined like the Levenshtein distance but adds transpositions as an edit operation of cost 1.

***n*-gram similarity** If we look at the set of n-grams of a string (either at the character or at the word level), then we can define a similarity between two strings as the Jaccard similarity of their sets of n-grams. (The Jaccard similarity of two sets is the size of their intersection divided by the size of their union.)

However, because we are dealing with large ontologies, the number of distinct literals is very high: for instance, there are 2.3 million distinct literals in DBpedia and 2.7 million in Yago. Hence, we cannot afford to compute the similarity between all pairs of literals. The algorithm for Paris suggests that the operation which we have to optimize is the following: given one literal x, we need to find the literals in the other ontology which are close to x for the similarity measure of interest.

A good idea to do so is to preprocess the literals of both ontologies and index them in a way which allows queries to be run more efficiently. This problem is called *approximate dictionary searching* and occurs in a variety of contexts: spell-checkers (which suggest dictionary words which are close to a misspelled word), automated speech recognition and optical character recognition (find dictionary words which are close to a recognized word), search engines (e.g., the "did you mean" feature of Google) and computational biology (find DNA sequences in a database which are similar to a query sequence).

Numerous methods are known for approximate dictionary searching within a certain distance threshold for the Levenshtein and Damerau–Levenshtein distances: see [Boy11] for a general survey. However, in our context, we do not need to respect an exact threshold for edit distance, but only want to get a reasonable number of good candidates (and do not require precision or recall to be 1). Besides, because we can both have small changes within words but also changes in word order, we can neither use the Levenshtein similarity at word level nor use it at character level. Instead, we choose to use the n-gram similarity measure at the character level.

The methods described in [Boy11] which index n-grams of a string are sequence-based filtering methods such as inverted n-gram files (section 7.1.2) or the unigram signature hashing (section 7.2.5). However, inverted n-gram file have a large index size which is problematic in our setting (because the index has to reside in main memory), and unigram signature hashing is a folklore method with poor performance. For these reasons, we turn to a different scheme.

6.3 Shingling

Shingling (or *MinHash* applied to the shingle set) is a technique which maps strings to a constant-size summary of their set of *shingles* (a.k.a. n-grams): these summaries can be indexed efficiently and an approximation of the Jaccard similarity of the shingle sets can be computed from them. It was introduced in [BGMZ97] to find duplicate in sets of Web pages, and presented in [MRS08, AMR$^+$11] for the same purpose, using as shingles the n-grams at word level. We use it to approximate the n-gram similarity measure on our literals, by taking as shingles the n-grams at character level. In this subsection, we review how this technique works and explain how we use it in our setting.

Shingling relies on the following observation: if we draw at random a hash function f from shingles to arbitrary values, then f essentially acts as a random permutation of the shingles. To simplify the analysis, we will assume that f has no collisions, which is extremely likely given the small number of possible shingles for the values of n that we use. Given a set of shingles S, the minimum $\phi_f(S) = \min_{s \in S} f(s)$ is thus realized by a random element of S under a uniform distribution; hence, given two sets S_1 and S_2 of Jaccard similarity $J(S_1, S_2)$, we have ([MRS08], p. 439 of the online edition):

$$\Pr_f(\phi_f(S_1) = \phi_f(S_2)) = J(S_1, S_2) \tag{14}$$

Now, notice that we can use the law of large numbers to get an arbitrarily good estimation of $J(S_1, S_2)$. Choose a constant N, and draw a family $(f_i)_{1 \leq i \leq N}$ of hash functions independently at random. Given a set of shingles $S = (s_j)$, we compute the $f_i(s_j)$ and define the *sketch* of S to be the N-uple $\psi(S) = (\min_j f_i(s_j))_{1 \leq i \leq N}$. As an abuse of notation, for two strings x and y, we will write $J(x, y)$ the Jaccard similarity of the set of shingles of x and y (which is the n-gram similarity of x and y), and will write $\psi(x)$ the sketch of the set of shingles of x. Remember that $L(x, y)$ is the n-gram similarity of x and y. Now, by Equation (14), if we consider two strings x and y, an unbiased estimator for the Jaccard similarity of the set of shingles of x and y is the proportion of positions at which the sketches of both strings are equal. Formally:

$$\lim_{N \to \infty} \frac{\left|\left\{i \in \{1, ..., N\} \mid \psi(x)_i = \psi(y)_i\right\}\right|}{N} = L(x, y)$$

The point of this estimator is that it can be used to index a collection of strings in an efficient fashion. Build N hash tables (H_i) (solving collisions by chaining), and for every string x of the collection, compute its sketch $\psi(x)$ and hash x in each hash table H_i at position $\psi(x)_i$. Of course, to save memory, we only store references to x in the hash tables, not copies of x.

Now, given a query string y and a threshold k, we can compute its sketch $\psi(y)$, and, from the N cells $(H_i[\psi(y)_i])_i$, extract those strings which occur at least k times. As an interesting side remark, note that the obvious algorithm to do this (scan the N cells, count the number of occurrences of each string, and keep those which occur more than k times) is not the most efficient one. The problem of retrieving the candidate strings is the *T-occurrence problem* of [LLL08] (section III, p. 2), for $T := k$. Their results show that elaborate approaches such as their DivideSkip algorithm can be significantly more efficient than the straightforward method (which they call ScanCount).

By the results above, the strings which occur in at least k of these N cells have been estimated to have a Jaccard similarity with y of at least k/N. Because this set of candidate strings should be small, we can actually afford to re-compute the exact similarity between them and the query string: the point of the estimator is only to filter the strings. Hence, by indexing the literals of one ontology, we can, given a query string from the other ontology, obtain efficiently the similar literals with their similarity scores.

6.4 Implementation

A standalone implementation of the shingling technique was implemented in Java by Mayur Garg who interned in the DBWeb team from IIT Delhi. Mayur also investigated the best choice of parameters for our needs: number of hash function to use, characteristics of the hash functions, and value of n for the n-grams, evaluated in terms of query time, memory usage, and recall. Details are available in his report [Gar12].

I helped supervise Mayur and assisted him in his work. I also improved the implementation of Paris to allow it to use Mayur's code.

6.5 Results

To investigate experimentally the efficiency of approximate literal matching, we tested the results of Paris with the shingling technique on the restaurant dataset from the Ontology Alignment Evaluation Initiative, 2010 [EFM$^+$10], which was used to evaluate the original version of Paris. We compare the shingling technique to the straightforward use of the exact equality function, and to the use of the exact equality function up to an ad-hoc normalization technique devised specifically for this dataset.

The results are summarized in Table 3 in Appendix B and show that the results of the shingling technique are comparable to ad-hoc normalization. By contrast, the shingling technique is generic and does not depend on the dataset.

7 Application to Deep Web Analysis and Ontology Enrichment

In this section, I present a possible use of Paris for information extraction from deep Web sources. This is joint work with Marilena Oita, who is currently doing her PhD on "Deriving Semantic Objects from the Structured Web" under Pierre Senellart's supervision. It has been accepted as a vision paper for the VLDS workshop of the VLDB conference [OAS12]. Parts of this section are directly adapted from the publication.

7.1 Introduction

The *deep Web* consists of dynamically-generated Web pages that are reachable by issuing queries through HTML forms. A form is a section of a document with special *control* elements (e.g., checkboxes, text inputs) and associated labels. Users generally interact with a form by modifying its controls (entering text, selecting menu items) before submitting it to a Web server for processing. Usually, the contents of the form are used to prepare a query which is run on a backend database, and the results of this query are presented on the response page.

Forms are primarily designed for human beings, but it is an important challenge to design automated agents that are able to use them. A first possible use of such agents is simply to crawl the result pages that can be obtained from the form: for instance, this is an improvement over existing search engine crawlers such as Googlebot which do not know how to fill forms and thus are not able to crawl and index result pages. Furthermore, more elaborate uses are possible thanks to the structured nature of the information on result pages: we can try to extract structured data from them, and use it to build ontologies or to enrich existing ontologies.

Most existing approaches for automated form understanding rely on domain knowledge: they assume that the domain of the form is known. Besides, they tend to separate the steps of form interface understanding and information extraction from result pages, although both contribute to a more *authentic* vision on the backend database schema. The form interface exposes in the input schema some attributes describing the query object, while response pages present this object instantiated in Web records that outline the form output schema.

A harder challenge is to understand the *semantics* of the backend database schema and how they relate to the object of the form. For instance, we want to understand that a given form allows us to query a database of books, that a certain input field and result page internal path corresponds to the title of the book, and that another field and internal path corresponds to the author of the book. Once again, previous attempts to achieve this have only relied on heuristics, or have assumed domain-specific knowledge.

Our approach to form interface understanding integrates two components. The first component is a system developed by Marilena Oita during her PhD which analyzes and probes forms, identifies records in result pages, establishes a mapping of the input and output schema, and represents the results of this crawling as an ontology. The second component is the Paris system, which is used to align this ontology to the general-purpose ontology Yago [SKW07]. This alignment allows us to identify the backend database objects (because the *entities* which represent them are aligned to Yago entities) and to understand the semantics of the form input fields and the output page internal paths (because the *relations* which represent them are aligned to Yago relations). This alignment crucially relies on new features of Paris that I developed during my internship: support for join relations (Section 4) and approximate literal matching (Section 6).

7.2 Building an Ontology from a Deep Web Source

The deep Web analysis system by Marilena produces a sample ontology from the deep Web source. In this section, I summarize how this is being done, by reviewing the first steps of Figure 6 in Appendix B which is an overview of the whole process. Details about the steps performed by Marilena's system are available in [OAS12].

The form is first *analyzed* and *probed* by entering stop words or neighboring terms into the form fields. The system *identifies records* from the result page using [OS12] and identifies the *output schema* as the DOM paths where record contents vary. It *aligns input and output schemas* by submitting record attribute values in form fields and trying to find matches.

Marilena's system then represents the data extracted from the Web records as RDF triples [W3C04d] in the following manner:

1. Each record is represented as an *entity*;
2. All records are of the same *class*, stated using `rdf:type`;
3. The attribute values of records are viewed as *literals*;
4. Each record links to its attribute values through the relation (i.e., *predicate*) that corresponds to the record internal path of the attribute type in the response page;

It should be noted that since this information is represented as RDF, we could add much more information to the representation provided that we have the means to extract it. Though Marilena's current implementation does not do this, we could include a more detailed representation of a record by following the hyperlinks that we identify in its attribute values and replacing them in the original response page with the DOM tree of the linked page. In this way, the extraction can be done on a more complete representation of the backend database. We could also add complementary data from various sources, e.g., Web services or other Web forms belonging to the same domain.

7.3 Ontology Alignment

The ontology generated in the previous section must now be aligned to a reference ontology. Paris is a good choice to perform this alignment, because it is holistic and we benefit from all the alignments that it is able to perform:

Entity alignment. This allows us to identify records from the deep Web source that are also present in the reference ontology.

Class alignment. This allows us to determine the class(es) of the deep Web records in the reference ontology.

Relation alignment. This allows us to determine the relations between the deep Web records, their realizations in the output schema of the form (e.g., understand that a certain DOM path in records indicates the author of the book represented by this record), and their realizations in the input schema of the form (e.g., understand that a certain form field allows us to search books by title).

The use of approximate literal matching is required to perform this alignment efficiently. Indeed, extracted literals usually differ from those of Yago because of alternate spellings or surrounding stop words. A typical case on Amazon is the addition of related terms, e.g., "Hamlet (French Edition)" instead of just "Hamlet". An independent possible improvement that could be made to the extraction phase to mitigate this issue is to perform pattern identification among data values of the same type.

Furthermore, the support for join relations is also necessary in this context, because attributes in the result pages of the form might not map to a simple relation in the source ontology. For instance, the "author" attribute in a form would correspond to a two-step path in Yago (`y:created`, `y:hasPreferredName`). To ensure that the alignment with joins can be performed efficiently, we limit the length of joins to 2. Since the sample ontology extracted from the deep Web source is small, this ensures that we will not explore the full reference ontology, but only those parts which are at a distance of at most 2 to a literal which aligned with a literal from the sample ontology.

Of course, a limitation of this whole approach is that there must be some overlap between the records of the deep Web source and the entities of the reference ontology. If this is not the case, then the alignment computed by Paris will necessarily be empty (or meaningless). If we use Yago as a reference ontology, then this means that we are limited to domains on which Wikipedia has sufficient coverage.

7.4 Form Understanding and Ontology Enrichment

Ontology alignment gives us knowledge about the data types, the domains and ranges of record attributes, and their relation to the object of the form (in our case, a book). The propagation of this knowledge to the input schema through the input–output mapping (for the form elements that have been successfully mapped) results in a better understanding of the form interface. On the one hand, we can infer that a given field of the Amazon advanced search form expects author names, and leverage Yago to obtain representative author names to fill in the form. This is useful in intensional or extensional automatic crawl strategies of deep Web sources. On the other hand, we can probe the interface to generate *new* result pages for which data location patterns are already known and enrich Yago through the alignment that we once determined.

There are three main possibilities to enrich the ontology. First, we can add to the ontology the *instances* that did not align (e.g., we can use the Amazon book search results to add to Yago the books for which it has no coverage). Second, we can add *facts* (triples) that were missing in Yago. Third, we can add the *relation types* that did not align. For instance, we can add information about the publisher of a book to Yago. This latter direction is more challenging, because we need to determine if the relation types contain valuable information. One safe way to deal with this *relevance problem* is to require attribute values to be mapped to a form element in the input schema. We can then use the label of the element to annotate them.

7.5 Results on Amazon Book Search

We have prototyped this approach for the Amazon book advanced search form[2]. Obviously, we cannot claim any statistical significance of the results we report here, but we believe that the approach, because it is generic, can be successfully applied to other sources of the deep Web.

Our preliminary implementation performed agnostic probing of the form, wrapper induction, and mapping of input–output schemas. It generated a labeled graph with 93 entities and 10 relation types out of which 2 (title and author) are recognized by Yago. Literals underwent a semi-heuristic normalization process (lowercasing, removal of parenthesized substrings). We then replaced each extracted literal with the most similar literal in Yago if the 2-gram similarity was higher than an arbitrary threshold.

We aligned this graph with Yago by running Paris for 15 iterations, i.e., a run time of 7 minutes (most of it was spent loading Yago, the proper computation took 20 seconds). Though the vast majority of the books from the dataset were not present in Yago, the 6 entity alignments with best confidence were books that had been correctly aligned through their title and author. To limit the effect of noise on relation alignment, we recomputed relation alignments on the entity alignments with highest confidence; the system was thus able to properly align the title and author relations with `y:hasPreferredName` and (`y:created`, `y:hasPreferredName`), respectively. These relations were associated to the record internal paths of the output schema attributes and propagated to form input fields.

Conclusion

Due to space constraints, we will not repeat the conclusion of our report here, but refer the reader to page 2 for a summary of contributions and a discussion of further work.

[2]`http://www.amazon.com/gp/browse.html?node=241582011`

Appendix A. References

[ADMR05] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 906–908, New York, NY, USA, 2005. ACM.

[Ama] Antoine Amarilli. Graph labeling schemes. `http://a3nm.net/work/research/graph_labeling_schemes.pdf`.

[AMR$^+$11] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management*, chapter 13. Cambridge University Press, 2011.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the Web. *Computer Networks and ISDN Systems*, 29(8–13):1157–1166, 1997. Papers from the Sixth International World Wide Web Conference.

[BLK$^+$09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, September 2009.

[Boy11] Leonid Boytsov. Indexing methods for approximate dictionary searching: Comparative analysis. *Journal of Experimental Algorithmics*, 16, May 2011.

[Dat] Internet Movie Database. `http://www.imdb.com/`.

[EFM$^+$10] Jérôme Euzenat, Alfio Ferrara, Christian Meilicke, Juan Pane, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, and Cássia Trojahn dos Santos. Results of the Ontology Alignment Evaluation Initiative 2010. In *OM'10*, 2010.

[Gar12] Mayur Garg. Intern report. Télécom ParisTech, July 2012.

[GS10] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *Journal of the ACM*, 57(2), January 2010.

[HPUZ10] Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann. Some entities are more equal than others: statistical methods to consolidate Linked Data. In *Proceedings of the Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic*, NeFoRS2010, 2010.

[Kol05] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 61–75, New York, NY, USA, 2005. ACM.

[LD94] Nada Lavrac and Saso Dzeroski. *Inductive logic programming - techniques and applications.* Ellis Horwood series in artificial intelligence. Ellis Horwood, 1994.

[Lib] British Library. `http://www.bl.uk/bibliographic/datasamples.html`.

[Lin] Linked Data. `http://linkeddata.org/`.

[LJPD$^+$12] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. SiGMa: Simple Greedy Matching for Aligning Large Knowledge Bases. `http://arxiv.org/abs/1207.4525`, 2012.

[LLL08] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 257–266, Washington, DC, USA, 2008. IEEE Computer Society.

[LMC11] Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 529–539, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[LTLL09] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21:1218–1232, 2009.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*, chapter 19. Cambridge University Press, New York, NY, USA, 2008.

[NM11] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 984–994, Washington, DC, USA, 2011. IEEE Computer Society.

[NW08] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowement*, 1(1):647–659, 2008.

[OAS12] Marilena Oita, Antoine Amarilli, and Pierre Senellart. Cross-fertilizing deep Web analysis and ontology enrichment. In *Proceedings of the 2nd International Workshop on Searching and Integrating New Web Data Sources*, VLDS '12, Istanbul, Turkey, 2012. Vision article.

[OS07] Yann Ollivier and Pierre Senellart. Finding related pages using Green measures: an illustration with Wikipedia. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1427–1433. AAAI Press, 2007.

[OS12] Marilena Oita and Pierre Senellart. Forest: Focused Object Retrieval by Exploiting Significant Tag paths. `http://pierre.senellart.com/publications/oita2013forest.pdf`, August 2012. Preprint.

[RD06] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

[SAS11] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. Paris: Probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment*, 5(3):157–168, 2011. Presented at the VLDB 2012 conference, Istanbul, Turkey.

[SKW07] 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.

[SSG$^+$] Fabian M. Suchanek, Pierre Senellart, Mayur Garg, Antoine Amarilli, and Serge Abiteboul. PARIS website. `http://webdam.inria.fr/paris/`.

[tCDK12] Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 182–195, New York, NY, USA, 2012. ACM.

[UGM07] Octavian Udrea, Lise Getoor, and Renée J. Miller. Leveraging data and structure in ontology integration. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 449–460, New York, NY, USA, 2007. ACM.

[W3C04a] W3C. RDF Test Cases. W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/`.

[W3C04b] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[W3C04c] W3C. RDF/XML Syntax Specification (Revised). W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.

[W3C04d] W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[W3C09] W3C. OWL 2 Web Ontology Language. W3C Recommendation, 2009. `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`.

[W3C11] W3C. Notation3 (N3): A readable RDF syntax. W3C Team Submission, 2011. `http://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/`.

[W3C12] W3C. Turtle: Terse RDF Triple Language. W3C Working Draft, 2012. `http://www.w3.org/TR/2012/WD-turtle-20120710/`.

[WM] Daniel S. Wilkerson and Scott McPeak. Delta. `http://delta.tigris.org/`.

[Zai10] Katrin Zaiß. *Instance-based ontology matching and the evaluation of matching systems*. PhD thesis, Universität Düsseldorf, Universitäts-und Landesbibliothek, 2010.

Appendix B. Tables and Figures

	Instances			Classes			Relations		
	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F
OAEI person	100%	100%	100%	100%	100%	100%	100%	100%	100%
OAEI restaurant	95%	88%	91%	100%	100%	100%	100%	66%	88%
DBpedia–YAGO	90%	73%	81%	94%	-	-	93%	-	-
IMDb–YAGO	94%	90%	92%	28%	-	-	100%	80%	89%

Table 1: Summary of the results of PARIS on the various datasets. Details are provided in [SAS11].

Iteration	**Original PARIS**	**New PARIS (1 thread)**	**New PARIS (4 threads)**
Startup	0h00	0h27	0h10
1	4h04	0h40	0h27
2	5h06	3h00	1h02
3	5h00	0h34	0h24
4	5h30	0h29	0h16
Total	20h	5h	2h

Table 2: Running times in hours for the DBpedia–YAGO alignment task. The original PARIS was run on an Intel Xeon E5620 CPU clocked at 2.40 Ghz on a machine with 12 GB of RAM. The new PARIS was run on an Intel Core i7-3820 CPU clocked at 3.60 Ghz with 48 GB of RAM.

	Precision	**Recall**	**F-measure**
PARIS with exact equality	95%	88%	91%
PARIS with shingling	96%	95%	96%
PARIS with normalization	98%	96%	97%

Table 3: Results of entity alignment on the restaurants OAEI dataset with exact equality, shingling, and ad-hoc normalization.

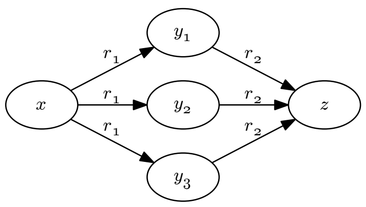
$$\begin{tikzcd} & y_1 \arrow[dr, "r_2"] & \\ x \arrow[ur, "r_1"] \arrow[r, "r_1"] \arrow[dr, "r_1"'] & y_2 \arrow[r, "r_2"] & z \\ & y_3 \arrow[ur, "r_2"'] & \end{tikzcd}$$

Figure 4: Because we project on the extremities, the join relation (r_1, r_2) only has one fact $(r_1, r_2)(x, z)$, not three, so we must store information to discard the duplicates and not count this fact three times by mistake.

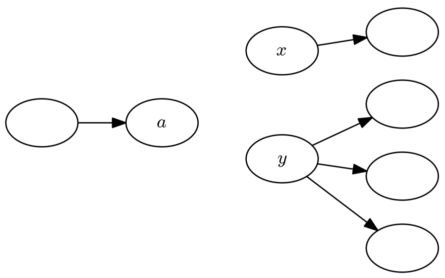

Figure 5: If we join the relation on the left (one fact) to the relation on the right (four facts), then the join relation can be either functional (if x and a are the same node) or not functional (if y and a are the same node).

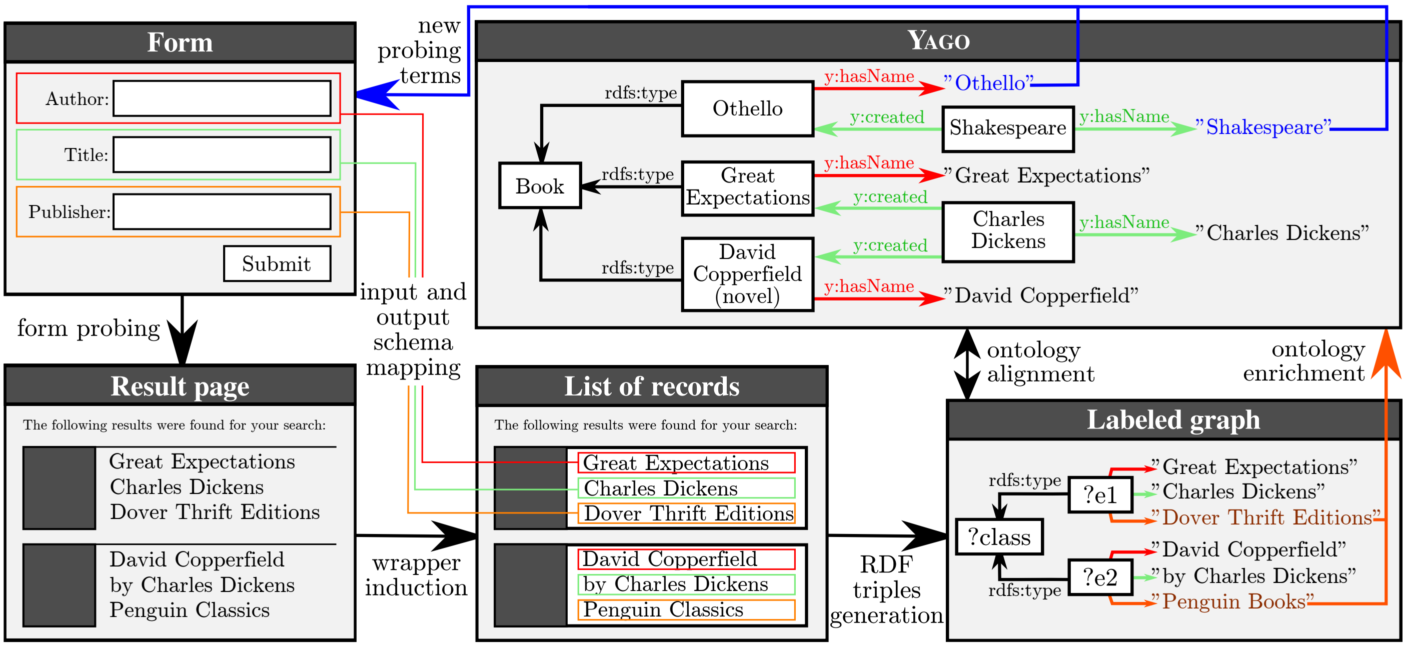

Figure 6: Overview of the envisioned approach

Appendix C. Refinements to the Entity Alignment Equation

In this appendix, I outline possible changes to Equation (1) by refining its probabilistic interpretation.

The propagation of entity alignment scores is given by Equation (1) on page 8. A condition that we would like to impose on this equation is that it should not lead to *evidence amplification*: namely, for any two entity couples (x, x') and (y, y'), if $P(y \equiv y')$ increases by ϵ, then $P(x \equiv x')$ should not increase by more than ϵ.

However, it turns out that this condition is not respected. In the simple case where the only nodes in the graph are x, x', y, and y', and where the only facts are $r(x, y)$ and $r'(x', y')$ where r and r' are two perfectly aligned and inverse functional relations (i.e., $P(r \subseteq r') = P(r \subseteq r') = 1$ and $\mathit{fun}^{-1}(r) = \mathit{fun}^{-1}(r') = 1$), Equation (1) rewrites as:

$$\Pr(x \equiv x') = 1 - \big(1 - \Pr(y \equiv y')\big)^2 \tag{15}$$

The derivative of $x \mapsto 1 - (1 - x)^2$ in 0 evaluates to 2, so, whenever $\Pr(y \equiv y')$ increases from 0 to a small value ϵ, the value of $\Pr(x \equiv x')$ increases by $2\epsilon + o(\epsilon)$. To fix this problem, a possible way would be to take the harmonic mean of the score from both directions, though this means that we lose the probabilistic interpretation (disjunction) that we had for the previous equation:

$$\Pr(x \equiv x') = 1 - \prod_{\substack{r(x,y)\\ r'(x',y')}} \sqrt{\big(1 - \Pr(r' \subseteq r) \times \mathit{fun}^{-1}(r) \times \Pr(y \equiv y')\big) \times \big(1 - \Pr(r \subseteq r') \times \mathit{fun}^{-1}(r') \times \Pr(y \equiv y')\big)} \tag{16}$$

Furthermore, another problem arises whenever multiple relations exist between two nodes. This situation is not unusual and occurs frequently in practice: for instance, in the YAGO ontology, a person can be related to the same place through `y:wasBornIn` and `y:diedIn`. As a simple example of the problem, let us assume that the facts are $r_1(x, y)$, $r_2(x, y)$, $r_1'(x, y)$, and $r_2'(x, y)$, that all relations are inverse functional, and that r_1 and r_1', r_2 and r_2' align perfectly in both directions ($\Pr(r_i \subseteq r_i') = \Pr(r_i' \subseteq r_i) = 1$ for $i = 1, 2$). See Figure 7 for an illustration. Substituting this in Equation (16), we obtain:

$$\Pr(x \equiv x') = 1 - \left(\sqrt{(1 - \Pr(y \equiv y'))^2}\right)^2 = 1 - \big(1 - \Pr(y \equiv y')\big)^2 \tag{17}$$

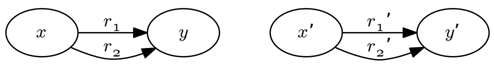

Figure 7: Multiple relations between entities.

We have the same problem as before because, intuitively, the evidence for $y \equiv y'$ is propagated once for each of the relation couples (r_1, r'_1) and (r_2, r'_2). The reason why this happens is because of the independence assumption: when we reach (y, y') by both relation couples, we consider that $\Pr(y \equiv y')$ is independent of itself, and take it into account twice. Indeed, consider the following logical formula, which is the justification of Equation (1) (where "i.f." stands for "inverse functional"):

$$\exists r, r', y, y',\ r(x, y) \wedge r'(x', y') \wedge y \equiv y' \wedge \big(\big(r \text{ is i.f.} \wedge r' \subseteq r\big) \vee \big(r' \text{ is i.f.} \wedge r \subseteq r'\big)\big) \Longrightarrow x \equiv x' \tag{18}$$

Equation (1) is obtained by translating (18) according to the process outlined in Appendix B of [SAS11]. However, this process depends on how we group the existential quantifiers. Translating Equation (18) by grouping the existential quantifiers as "$\exists(r, r', y, y')$" yields Equation (1). However, if we choose to group them as "$\exists(y, y')$, $\exists(r, r')$", we can rewrite the formula as:

$$\begin{gathered}\exists y, y', R, R',\ R(x, y) \wedge R'(x', y') \wedge F(R, R') \wedge y \equiv y' \Longrightarrow x \equiv x' \\ F(R, R') := \exists r \in R, r' \in R',\ \big(r \text{ is i.f.} \wedge r' \subseteq r\big) \vee \big(r' \text{ is i.f.} \wedge r \subseteq r'\big)\end{gathered} \tag{19}$$

This logical formula is equivalent to (18), but its reformulation accounts for the fact that there is no possible independence assumption between the various ways to reach a given couple (y, y') through several r's:

$$\Pr(x \equiv x') = 1 - \prod_{\substack{R(x,y) \\ R'(x',y')}} \big(1 - F(R, R') \times \Pr(y \equiv y')\big) \times \big(1 - F(R', R) \times \Pr(y \equiv y')\big) \tag{20}$$

$$(R, R') := 1 - \prod_{\substack{r \in R \\ r' \in R'}} \big(1 - \mathit{fun}^{-1}(r) \Pr(r' \subseteq r)\big) \tag{21}$$

The intuitive interpretation of this is that for a pair (x, x'), we do an independent disjunction over all adjacent pairs (y, y'), and x and x' are aligned if y and y' are aligned and if some couple of relations is suitable, which is the independent disjunction over all couple of relations that the relations are aligned and inverse functional.

This alternate way to align entities was implemented as an optional setting in PARIS. However, it does not lead to an observable difference in quality.

Appendix D. Counter-Example to Convergence with Loop Facts and TakeMax

In this appendix, I outline a simple counter-example to the convergence of PARIS if we allow facts of the form $r(x, x)$ and if we pick one maximal alignment for each entity at each step. The counter-example was discovered on real data and found by minification using `multidelta` [WM].

Consider the situation in Figure 8. At the first iteration, we will either align a and x_2, or a and y_2 (and only one of these choices because we pick one maximal alignment). Say we align a and x_2. Then, at the next iteration, the alignment $a \equiv x_2$ will make us align a and y_2. At the next iteration, this alignment will make us align again a and x_2. Thus, the alignment oscillates and does not converge.

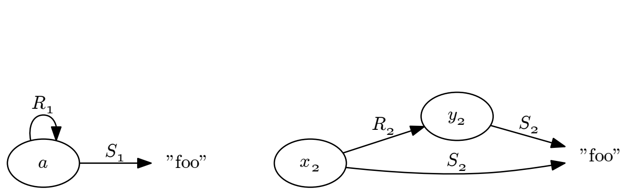

Figure 8: Example of non-convergence on two ontologies. We assume that the ontologies also contain data which encourage us to align R_1 and R_2, R_1^- and R_2, and S_1 and S_2.