Similarity-based ontology alignment in OWL-Lite

C0377

Abstract. Interoperability of heterogeneous systems on the Web will be admittedly achieved through an agreement between the underlying ontologies. However, the richer the ontology description language, the more complex the agreement process, and hence the more sophisticated the required tools. Among current ontology alignment paradigms, similarity-based approaches are both powerful and flexible enough for aligning ontologies expressed in languages like OWL. We define a universal measure for comparing the entities of two ontologies that is based on a simple and homogeneous comparison principle: Similarity depends on the type of entity and involves all the features that make its definition (such as superclasses, properties, instances, etc.). One-to-many relationships and circularity in entity descriptions constitute the key difficulties in this context: These are dealt with through local matching of entity sets and iterative computation of recursively dependent similarities, respectively.

1 The ontology alignment problem

Ontologies are seen as the solution to data heterogeneity on the web. However, the available ontologies could themselves introduce heterogeneity: given two ontologies, the same entity can be given different names or simply be defined in different ways, whereas both ontologies may express the same knowledge but in different languages.

Semantic interoperability can be grounded in ontology reconciliation. The underlying problem, which we call the "ontology alignment" problem, can be described as follows: given two ontologies each describing a set of discrete entities (which can be classes, properties, predicates, etc.), find the relationships (e.g., equivalence or subsumption) that hold between these entities. Alignment results can further support visualization of correspondences, transformation of one source into another or formulation of bridge axioms between the ontologies. An overview of alignment methods is presented in §2.

We focus on automatic and autonomous ontology alignment, although more interactive scenarios may be built on top of the proposed technique (e.g., complete a partial alignment or use the result as a suggestion to the user). It will be also assumed that the ontologies are described within the same knowledge representation language: OWL-Lite (§3.1). The language is based on various features: classes and subsumption, properties and type constraints, etc. Instead of using the external textual form, we define a dedicated typed graph representation of the language that concentrates the necessary information for computing the similarity between OWL entities (§3.2).

This paper defines a similarity measure that encompasses all OWL-Lite features (§4.1) while overcoming key alignment problems such as collection comparison (§4.2) and circularities (§4.3). Our approach is based on previous work on object-based knowledge representation similarity [15] which is here adapted to the alignment within current web languages. The computed measure is then used for generating an actual alignment (§5).

2 Alignment methods

There has been important background work that can be used for ontology alignment: in discrete mathematics for matching graphs and trees [7, 11], in databases for reconciling and merging schemas [12], in machine learning for clustering compound objects described in a restricted FOL [1].

Basically, aligning amounts at defining a distance between entities (which can be as reduced as an equality predicate) and computing the best match between ontologies, i.e., the one that minimizes the total distance (or maximizes a similarity measure). But there are many different ways to compute such a distance. Roughly speaking, they can be classified as (this complements the taxonomy provided in [12] and only considers features found in actual systems):

terminological (T) comparing the labels of the entities; **string-based (TS)** does the terminological matching through string structure dissimilarity (e.g., edit distance); **terminological with lexicons (TL)** does the terminological matching modulo the relationships found in a lexicon (i.e., considering synonyms as equivalent and hyponyms as subsumed);

internal structure comparison (I) comparing the internal structure of entities (e.g., the value range or cardinality of their attributes);

external structure comparison (S) comparing the relations of the entities with other entities; **taxonomical structure (ST)** comparing the position of the entities within a taxonomy; **external structure comparison with cycles (SC)** an external structure comparison robust to cycles;

extensional comparison (E) comparing the known extension of entities, i.e. the set of other entities that are attached to them (in general instances of classes);

semantic comparison (M) comparing the interpretations (or more exactly the models) of the entities.

Some contributions are listed in Table 1, we only provide some salient points for each of them: [3] matches conceptual graphs using terminological linguistic techniques and comparing superclasses and subclasses. [13] computes the dissimilarity between two taxonomies by comparing for each class the labels of their superclasses and subclasses. FCA-Merge [14] uses formal concept analysis techniques to merge two ontologies sharing the same set of instances while properties of classes are ignored. Anchor-Prompt [10] uses a bounded path comparison algorithm with the originality that anchor points can be provided by the users as a partial alignment. Cupid [8] is a first approach combining many of the other techniques. It aligns acyclic structures taking into account terminology and data types (internal structure) and giving more importance to leaves. [9] creates a graph whose nodes are candidate aligned pairs and arcs are shared properties. Arcs are weighted by their relevance to nodes and similarity values are propagated through this graph in a search for a fixed point.

T-tree [5] infers dependencies between classes (bridges) of different ontologies sharing the same set of instances based only on the "extension" of classes. The algorithm of [6] uses a complete prover to decide subsumption or equivalence between classes given initial equivalence of some classes and analysis of the relationships in the taxonomy. In addition, a number of other systems use machine learning techniques for finding class similarity from instances [4].

Reference	T	TS	TL	I	S	ST	SC	E	M
Dieng & Hug [3]	x		x			x			
Staab & Mädche [13]		x			x	x			
FCA-Merge [14]						x		x	
Prompt [10]	x			x	x	x			
Cupid [8]	x	x		x	x				
Sim. flooding [9]		x			x		x		
T-tree [5]					x			x	
Sem. match [6]						x			x

Table 1. Various contributions to alignment at a glance.

Despite the variety of alignment techniques used, most of the above methods cover only subsets of the ontology definitions and are not robust enough to cycles in definitions, e.g., the iterative computation in [9] need not to converge. Our goal is to design a measure that integrates all aspects of OWL-Lite while successfully dealing with cyclic definitions.

3 Ontology representation

To that end, we designed a representation for OWL-Lite ontologies (§3.1) that emphasizes entities and their relationships (§3.2).

3.1 The web ontology language OWL

OWL [2] is a language for expressing ontologies on the web. Due to space restrictions, we only present here the ontology constructors proposed by the language (the reader can find elsewhere more information on their semantics). OWL can be thought of as a description logic embedded in a frame-like syntax. It comes in three flavors: OWL-Lite, OWL-DL, and OWL-Full. We concentrate on OWL-Lite which is sufficient for many purposes while creating various difficulties for alignment algorithms.

OWL-Lite is an extension of RDF which allows the definition of individuals as class instances and the characterization of inter-class relations (like in Figure 1). Additionally, OWL-Lite:

- uses RDF Schema keywords (`rdfs:subClassOf`, `rdfs:Property`, `rdfs:subPropertyOf`, `rdfs:range`, `rdfs:domain`) for defining taxonomies of classes and properties and restricting the range of properties;
- allows the definition of a class (`owl:Class`) as more specific or equivalent to the intersection of other classes;
- allows the assertion of equality (`owl:sameAs`) or difference (`owl:differentFrom`) between two individuals;
- characterizes properties as inverse (`owl:inverseOf`), transitive (`owl:TransitiveProperty`) or symmetric (`owl:SymmetricProperty`);
- can restrict the range of a property in a class to be another class (`owl:allValuesFrom`) or assert that some objects of a particular class must be in the property (`owl:someValuesFrom`).
- can restrict the number of objects in a particular relation with another one through the use of cardinality constraints (`owl:minCardinality` and `owl:maxCardinality`). In OWL-Lite, these constraints can only take values 0, 1, or infinite.

This is enough for expressing complex class expressions as those in the UML diagram of Figure 1.

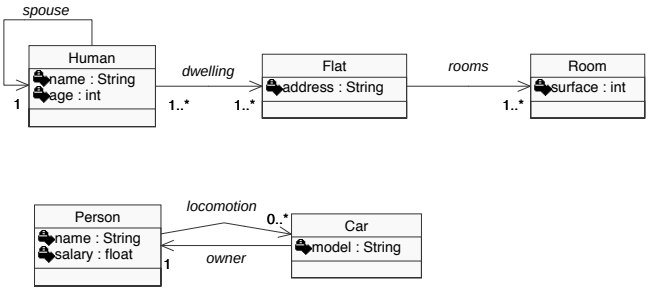

Figure 1. Classes of two ontologies drawn as UML class diagrams.

OWL makes use of external data types. In particular it relies on XML Schema datatypes.

3.2 Representation

When it comes to comparing descriptions that encode relational information about the represented entities, as OWL does, the textual form may easily prove much too rigid. Conversely, the graphic representation allowed by the RDF syntax is too flexible as it provides many different ways to express the same constraint and does not provide typing information without traversing the graph. Thus, a corresponding but much more explicit graph-based syntax is used here. The OL-graphs, as we call them, contain the following categories of nodes: class (C), object (O), relation (R), property (P), property instance (A), datatype (D), datavalue (V), property restriction labels (L). OL-graph edges model various relationships:

- `rdfs:subClassOf` between two classes or two properties ($\mathcal{S}$);
- `rdf:type` ($\mathcal{I}$) between objects and classes, property instances and properties, values and datatypes;
- $\mathcal{A}$ between classes and properties, objects and property instances;
- `owl:Restriction` ($\mathcal{R}$) expressing the restriction on a property in a class;
- valuation ($\mathcal{U}$) of a property in an individual

In the remainder relation symbols are used as set-valued functions ($\mathcal{F}(x) = \{x; \exists y; \langle x, y\rangle \in \mathcal{F}\}$). Additionally, each node is identified ($\lambda : C \cup O \cup R \cup P \cup D \cup A \longrightarrow URIRef$) by a URI reference and can be attached annotations.

The graph structure makes relationships between language elements more explicit, e.g., if a class c refers to c' via a `owl:allValuesFrom` restriction, a path between the corresponding nodes in the OL-graph will occur. OL-graphs record further information that can support comparison, e.g., descriptive knowledge inherited from nodes of the same or related categories. Finally, to provide the most complete basis for comparison, one may wish to bring knowledge encoded in relation types to the object level. This could be done by adding edges between objects that are reverse, symmetric or transitive for an existing edge or a pair of

edges. Relation types can be handled by saturation of the graph or in a lazy way: for `owl:TransitiveProperty` by adding transitivity arcs; for `owl:SymmetricProperty` by adding symmetric arcs; for `owl:inverseOf` by adding the reverse arcs (both in generic and individual descriptions); for `owl:FunctionnalProperty` by adding a cardinality constraint; `owl:InverseFunctionnalProperty` is not accounted for at that stage.

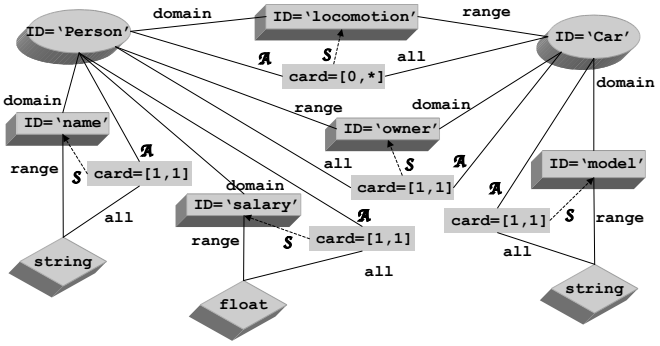

Figure 2. The OL-graph of the second set of classes from Figure 1.

To sum up, a node characterization is mostly spread over its adjacent edges. Thus, since each link of a given category connects two nodes of precisely defined categories, each node description may be seen as a fixed collection of homogeneous sets of links.

4 Similarity of OWL entities

Building on the idea of regularity in node characterizations within a OL-graph, we establish a family of similarity measures (§4.1), one per node category, whose mutual dependencies follow the structure of the graph which in turn reflects the OWL grammar. Processing of node/link sets (§4.2) and resolution of recursive dependencies in similarity definition (§4.3) are key elements of our model.

4.1 Similarity measure definition

The similarity (Sim_X) defined between nodes of the OL-graphs follows two principles: (i) it depends on the category X of node considered and (ii) it takes into account all the features of this category (e.g., superclasses, properties).

When similarity is sought, identically labeled edges of a type $\mathcal{F}$, from two compared nodes n_1 and n_2 (of category X) induce a dependency for the similarity $Sim_X(n_1, n_2)$ on the similarity of each pair of nodes (n'_1, n'_2) such that $n'_i, \in \mathcal{F}(n_i)$ $(i = 1, 2)$. Indeed, the higher the similarity of every single (n'_1, n'_2), the higher $Sim_X(n_1, n_2)$. The pair whose similarity is under assessment, here (n_1, n_2), will be further referred to as the *anchor* pair of the comparison and all the pairs (n'_1, n'_2) as the *contributor* ones (since the similarity of (n'_1, n'_2) *potentially* contributes to $Sim_X(n_1, n_2)$. Although the contributor relationship could be transitively spread throughout a pair of OL-graphs by inductively defining contributors of order n for $n = 1, 2, \ldots$, we only consider the direct, or 1st-order, contributors and 0th-order ones, i.e., those lying within n_1 and n_2 like their respective labels or URI references.

Given two nodes n_1 and n_2 from X, $Sim_X(n_1, n_2)$ is a function of the similarities of the 0th- and 1st-order contributors. Moreover, an anchor pair of a particular computation may be a contributor in another one with a specific measure used in each case. However, for homogeneity reasons, we require a particular node pair to always be assigned the same similarity value.

Finally, for computational reasons that will be made clear later on, a linear (weighted) combination for all similarity functions has been chosen. It is widely accepted that weights are useful in such cases to control the contribution of each factor[1]. Formally, given a category X its similarity measure $Sim_X : X^2 \to [0, 1]$ is as follows:

$$Sim_X(x, x') = \sum_{\mathcal{F} \in \mathcal{N}(X)} \pi^X_{\mathcal{F}} MSim_Y(\mathcal{F}(x), \mathcal{F}(x'))$$

where $\mathcal{N}(X)$ is the set of all relationships in which X participates (see §3.2). The weights $\pi^X_{\mathcal{F}}$ are normalized, i.e., $\sum_{\mathcal{F} \in \mathcal{N}(X)} \pi^X_{\mathcal{F}} = 1$ (see §4.3). For instance, for two classes c, c' :

$$\begin{aligned} Sim_C(c, c') = {} & \pi^C_L sim_L(\lambda(c), \lambda(c')) \\ & + \pi^C_{\mathcal{I}} MSim_O(\mathcal{I}(c), \mathcal{I}'(c')) \\ & + \pi^C_{\mathcal{S}} MSim_C(\mathcal{S}(c), \mathcal{S}'(c')) \\ & + \pi^C_{\mathcal{A}} MSim_P(\mathcal{A}(c), \mathcal{A}'(c')) \end{aligned}$$

The set functions $MSim_Y$, where the category Y depends on the relationship $\mathcal{F}$, are presented in §4.2. Thus, the similarity of `Flat` and `Person` is:

$$\begin{aligned} Sim_C(\texttt{Flat}, \texttt{Person}) = {} & \pi^C_L sim_L(\texttt{'flat'}, \texttt{'person'}) \\ & + \pi^C_{\mathcal{A}dt} MSim_P(\{\texttt{rooms}\}, \{\texttt{locomotion}\}) \\ & + \pi^C_{\mathcal{A}o} MSim_P(\{\texttt{address}\}, \{\texttt{name}, \texttt{salary}\}) \end{aligned}$$

where $\pi^C_{\mathcal{A}dt}$ and $\pi^C_{\mathcal{A}o}$ are the components of $\pi^C_{\mathcal{A}}$ assigned to data type and object property parts of $Sim_C(\texttt{Flat}, \texttt{Person})$, respectively (no $\pi^C_{\mathcal{F}}$ means $\mathcal{F}$ contributors are ignored). Table 2 illustrates the set of similarities in our model.

Funct.	Node	Factor	Measure
Sim_O	$o \in O$	$\lambda(o)$	sim_L
		$a \in A, (o, a) \in \mathcal{A}$	$MSim_A$
Sim_A	$a \in A$	$r \in R, (a, r) \in \mathcal{R}$	Sim_R
		$o \in O, (a, o) \in \mathcal{U}$	$MSim_O$
		$v \in V, (a, v) \in \mathcal{U}$	$MSim_V$
Sim_V	$v \in V$	value literal	type dependent
Sim_C	$c \in C$	$\lambda(c)$	sim_L
		$p \in P, (c, p) \in \mathcal{A}$	$MSim_P$
		$c' \in C, (c, c') \in \mathcal{S}$	$MSim_C$
sim_D	$d \in D$	$\lambda(r)$	XML-Schema
Sim_R	$r \in R$	$\lambda(r)$	sim_L
		$c \in C, (r, \texttt{domain}, c) \in \mathcal{R}$	$MSim_C$
		$c \in C, (r, \texttt{range}, c) \in \mathcal{R}$	$MSim_C$
		$d \in D, (r, \texttt{range}, d) \in \mathcal{R}$	Sim_D
		$r' \in R, (r, r') \in \mathcal{S}$	$MSim_R$
Sim_P	$p \in P$	$r \in R, (p, r') \in \mathcal{S}$	Sim_R
		$c \in C, (p, \texttt{all}, c) \in \mathcal{R}$	$MSim_C$
		$n \in \{0, 1, \infty\}, (p, \texttt{card}, n) \in \mathcal{R}$	equality

Table 2. Similarity function decomposition (`card` = `cardinality` and `all` = `allValuesFrom`).

Although only 0th- and 1st-order contributors are included in similarity definition, its recursive nature allows higher-order contributors to impact an anchor pair similarity as well[2]. In other terms, we let

[1] While weight assignment knowingly requires non-trivial knowledge about the domain of the compared data, this drawback is attenuated here by linking weights to entire descriptive aspects instead of particular entity features.

[2] However their real contribution is inversely propositional to their order.

the similarity "current" flow, or flood as in [9], through edges linking contributor and anchor pairs. The target similarity values ultimately depend on the comparison of data types values, and on the way its results propagate throughout the graphs. Measures for data types and values should be provided together with an abstract data type definition, URI references can be compared by an equality predicate or by a string similarity applied to suffixes.

4.2 Similarity-based matching of entity collections

Given an anchor pair n_1 and n_2 from X, and a particular link type $\mathcal{F}$, $\mathcal{F}(n_i) = S_i$ $(i = 1, 2)$ is generally a set of nodes, hence the set of contributors $S_1 \times S_2$. In order to ensure balance between different factors in $Sim_X(n_1, n_2)$, all contributor similarities are combined into a single value, by a set similarity function $MSim$. For normalization reasons, all $MSim$ functions are averages that range over a restricted subset of $S_1 \times S_2$ which is chosen in a way that ensures consistency and relevance. Therefore, it could be seen as an assignment of 0/1 weights to the members of $S_1 \times S_2$. Formally, a matching $Pairing(S, S')$ of both sets is established which is: (i) of maximal total similarity, (ii) exclusive, and (iii) of maximal size [15]. The value of $MSim_C(S, S')$ is simply the total similarity of the contributors in $Pairing(S, S')$, divided by the size of the larger set:

$$MSim_C(S, S') = \frac{\sum_{\langle c,c' \rangle \in Pairing(S,S')} Sim_C(c, c')}{max(|S|, |S'|)}$$

It is noteworthy that the matching at each anchor pair remains local, i.e., it has no impact on the global ontology alignment which is also a specific case of matching. In fact, matching a pair of nodes, i.e., choosing it as an effective contributor to another one's similarity only implies that the underlying resemblances are high for the local context, but not necessarily for the entire ontology.

To illustrate $MSim$, consider the example of §4.1 and take the data type factor of the similarity $Sim_C(\texttt{Flat}, \texttt{Person})$. Assume also that $Sim_P(\texttt{address}, \texttt{name}) = 0.64$ and $Sim_P(\texttt{address}, \texttt{salary}) = 0.34$ which means the best matching is $\{(\texttt{address}, \texttt{name})\}$. Thus,

$$MSim_P(\{\texttt{address}\}, \{\texttt{name}, \texttt{salary}\}) = 0.64/2 = 0.32.$$

4.3 Effectively computing similarities

It may easily happen that two pairs of nodes are each other's contributor, e.g., $(\texttt{Flat}, \texttt{Person})$ and $(\texttt{rooms}, \texttt{locomotion})$. The resulting recursive dependency of $Sim_C(\texttt{Flat}, \texttt{Person})$ on $Sim_P(\texttt{rooms}, \texttt{locomotion})$ and *vice versa* requires non standard computation means. As shown by Bisson [1], an equation system may be composed where the target similarity values are the solutions.

To that end, each pair of "alignable" nodes, i.e., in C, R and, possibly, O, is assigned a variable x_i, y_j, and z_k, respectively, that represents its similarity. An equation is composed by developing the denoted similarity along the guidelines of §4.1 while replacing contributor similarities by the corresponding variables. As an illustration, consider $Sim_C(\texttt{Flat}, \texttt{Person})$, and assume the following weights for C and P categories[3]:

π^C_L	$\pi^C_{\mathcal{I}}$	π^C_S	$\pi^C_{\mathcal{A}o}$	$\pi^C_{\mathcal{A}d}$	π^P_L	π^P_{card}	$\pi^P_{\mathcal{A}}$	$\pi^P_{\mathcal{R}}$
.4	0.	.1	.25	.25	.25	.15	.4	.2

[3] Space limitations force the merge of P and R categories in similarity computation. However, the only difference is that property name and domain are directly included in Sim_P instead of impacting it via Sim_R.

Variable substitutions are as follows:

$$\begin{aligned} x_1 &= Sim_C(\texttt{Flat}, \texttt{Person}) & x_2 &= Sim_C(\texttt{Room}, \texttt{Car}) \\ x_3 &= Sim_C(\texttt{Human}, \texttt{Person}) & x_4 &= Sim_C(\texttt{Flat}, \texttt{Car}) \\ x_5 &= Sim_C(\texttt{Room}, \texttt{Person}) & x_6 &= Sim_C(\texttt{Human}, \texttt{Car}) \\ y_1 &= Sim_R(\texttt{rooms}, \texttt{locomotion}) & y_2 &= Sim_R(\texttt{address}, \texttt{name}) \\ y_3 &= Sim_R(\texttt{address}, \texttt{salary}) & y_4 &= Sim_R(\texttt{surface}, \texttt{model}) \end{aligned}$$

In the following computation, datatype similarities are set to the identity function whereas cardinality measure is 1 if both limits correspond, 0.5 if only one does ($*$ is ignored), 0.35 if no match, but still there is inclusion, and 0 otherwise. Moreover, the following (arbitrary) label similarity values are assumed:

$$\begin{aligned} sim_L(\texttt{flat}, \texttt{person}) &= .4 & sim_L(\texttt{room}, \texttt{car}) &= .5 \\ sim_L(\texttt{rooms}, \texttt{locomotion}) &= .25 & sim_L(\texttt{address}, \texttt{name}) &= .7 \\ sim_L(\texttt{address}, \texttt{salary}) &= .3 & sim_L(\texttt{surface}, \texttt{model}) &= .35 \end{aligned}$$

The following equations are hence composed whereby $choice(S)$ simulates the set matching underlying S, i.e., it assigns 0/1 weights to each variable in the set:

$$\begin{aligned} x_1 &= .16 + .25 * choice(\{y_1\}) & &+ .125 * choice(\{y_2, y_3\}) \\ y_1 &= .115 + .4 * choice(\{x_1\}) & &+ .2 * choice(\{x_2\}) \\ x_2 &= .2 + .125 * choice(\{y_4\}) & y_2 &= .525 + .4 * choice(\{x_1\}) \\ y_3 &= .225 + .4 * choice(\{x_1\}) & y_4 &= .238 + .4 * choice(\{x_2\}) \end{aligned}$$

If each choice could be established a priori, i.e., regardless of effective similarity values, the result would be a directly solvable system (since linear). As OWL-Lite ontologies typically produce non-linear systems, choices cannot be established beforehand. Nevertheless, the resulting system could still be solved through an iterative process that produces the nearest reachable fixed point of a vector function [1]. The iteration produces a sequence of ever more precise approximations of the target solution vector. Initial similarity values are based only on 0-th level contributors. The values at step $n+1$ are then computed using the similarities of the 1st-level contributors from step n (including matching re-calculations).

The process of iterative resolution always converges to a solution. This could be easily proved by induction on the iteration steps: the sequences of values for each variable in the system are non-decreasing and bounded by 1, so they converge to a limit in $[0, 1]$. Moreover, the iterative resolution mechanism is flexible enough to admit user-provided similarity values (or dissimilarity assertions): these are used as constant values for the corresponding variable while the respective equation is ignored.

The following solutions for selected variables from the entire system corresponding to Figure 1 were found by the process in six steps:

$$\begin{array}{lllll} x_1 = .293 & x_3 = .566 & x_5 = .156 & y_1 = .290 & y_3 = .342 \\ x_2 = .288 & x_4 = .492 & x_6 = .370 & y_2 = .642 & y_4 = .353 \end{array}$$

5 Structure alignment as similarity maximizing

The result of the iterative process is not an alignment on its own, but rather an approximation of the similarity between entities from opposite ontologies. Nevertheless, the computed similarities suggest possible mappings of entities, hence they can support an alignment. One way of doing this consists in displaying the entity pairs with their similarity scores and/or ranks and leaving the choice of the appropriate pairs up to the user of the alignment tool.

One could go a step further and attempt at defining algorithms that automate alignment extraction from similarity scores. Various strategies may be applied to the task depending on the properties of the target alignment. As a matter of fact, one can ask the alignment to be complete (total) for one of the ontologies, i.e., all the entities of that ontology must be successfully mapped on the other side. Completeness is purposeful whenever thoroughly transcribing knowledge from one ontology to another is the goal. One can also require the mapping to be injective and hence reversible.

If neither ontology needs to be completely covered by the alignment, a threshold-based filtering would allows us to retain only the most similar entity pairs. Without the injectivity constraint, the pairs scoring above the threshold represent a sensible alignment. In contrast, if an injective mapping is required then some choices need to be made in order to maximize the "quality" of the alignement that is typically measured on the total similarity of the aligned entity pairs. Consequently, the alignment algorithm must optimize the global criteria rather than maximizing the local similarity at each entity pair.

To sum up, the alignment computation may seen as a less constrained version of the basic set similarity functions $MSim$. Indeed, its target features are the same: (i) maximal total similarity, (ii) exclusivity and (iii) maximal cardinality (in entity pairs). However, (ii) and (iii) are not mandatory, they depend on injectivity and completeness requirements, respectively.

A greedy alignment algorithm could construct the correspondances step-wise, at each step selecting the most similar pair and deleting its members from the table. The algorithm will then stop whenever no pair remains whose similarity is above the threshold.

The greedy strategy is not optimal: finding the global optimum would require the computing of a square assignment (polynomial assignment algorithms are suggested in [11]). However, the ground on which a high similarity is forgotten to the advantage of lower similarities can be questioned and thus the greedy algorithm could be preferred in some situations.

Given the results produced in the previous section, a threshold of .5 will select only the correspondence between `Human` and `Person`. There is no point in asking for a complete alignment here since the classes are quite different (the requirement of a maximal match would have `Car` associated to `Flat`). Concerning properties, the highest value of the entire system that (not given here) was measured on the pair of `name` properties, as expected. More interestingly, the chosen settings strongly suggest the identification of `address` to `model` and `age` to `salary`. This fact illustrates the difficulties in discriminating 1st-order contributors of an anchor pair having similar ranges.

6 Conclusion

To support the alignment of ontologies in OWL-Lite, we adapted a method that was initially designed for instance similarities in object-based languages. The new method has the advantage of incorporating most of the descriptive features of an ontology into the alignment computing process: it deals successfully with external data types, internal structure of classes as given by their properties and constraints, external structure of classes as given by their relationships to other classes and the availability of individuals. Moreover, new features as well as new datatypes can be accommodated through new categories of OL-graph nodes or new base similarity functions.

The resulting extensibility is a clear improvement upon other methods that take advantage only of a subset of all language features in OWL-Lite. The proposed method not only composes linearly individual means for assessing the similarity between entities, but it also provides an integrated similarity definition that makes those means interact during computation. Moreover, it successfully copes with the unavoidable circularities that occur within ontologies.

Yet, this measure does not cover all syntactic constructions of OWL-Lite (e.g., `owl:AllDifferent`, `owl:InverseFunctionnalProperty`). We plan to integrate them as well as some features of OWL-DL (e.g., `owl:oneOf`). Moreover, thorough tests of our measure must be performed to find weight ranges and external similarity measures that provide satisfactory results while favoring predictable behaviour.

Although the proposed similarity measure is not semantically justified, it exhibits valuable features such as fitting to various classes of inter-ontology mappings. Furthermore, we would like it to be at least syntax-independent for OWL-Lite. To that end, one must ensure that whatever the description of two entities, if they are semantically equivalent, they behave identically with respect to the similarity measure. This will amount to either normalizing the graph (e.g., adding `owl:minCardinality` constraints for each `owl:someValueFrom` for instance) or designing heterogeneous comparison means.

REFERENCES

[1] Gilles Bisson. Learning in FOL with similarity measure. In *Proc. 10th American Association for Artificial Intelligence conference, San-Jose (CA US)*, pages 82–87, 1992.

[2] Mike Dean and Guus Schreiber (eds.). OWL web ontology language: reference. Recommendation, W3C, 2004. http://www.w3.org/TR/owl-ref/.

[3] Rose Dieng and Stefan Hug. Comparison of "personal ontologies" represented through conceptual graphs. In *Proc. 13th ECAI, Brighton (UK)*, pages 341–345, 1998.

[4] An-Hai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Ontology matching: A machine learning approach. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag, Heildelberg (DE), 2003.

[5] Jérôme Euzenat. Brief overview of T-tree: the Tropes taxonomy building tool. In *Proc. 4th ASIS SIG/CR workshop on classification research, Columbus (OH US)*, pages 69–87, 1994. ftp://ftp.inrialpes.fr/pub/sherpa/publications/euzenat93c.ps.gz.

[6] Fausto Giunchiglia and Pavel Shvaiko. Semantic matching. In *Proc. IJCAI 2003 Workshop on ontologies and distributed systems*, pages 139–146, 2003.

[7] John Hopcroft and Robert Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[8] Jayant Madhavan, Philip Bernstein, and Erhard Rahm. Generic schema matching using Cupid. In *Proc. 27th VLDB, Roma (IT)*, pages 48–58, 2001. http://research.microsoft.com/ philbe/CupidVLDB01.pdf.

[9] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proc. 18th International Conference on Data Engineering (ICDE), San Jose (CA US)*, 2002.

[10] Natalya Noy and Mark Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proc. IJCAI 2001 workshop on ontology and information sharing, Seattle (WA US)*, pages 63–70, 2001. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-47/.

[11] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Prentice-Hall, 1998.

[12] Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[13] Steffen Staab and Alexander Mädche. Measuring similarity between ontologies. *Lecture notes in artificial intelligence*, 2473:251–263, 2002.

[14] Gerd Stumme and Alexander Mädche. FCA-merge: bottom-up merging of ontologies. In *Proc. 17th IJCAI, Seattle (WA US)*, pages 225–230, 2001.

[15] Petko Valtchev. *Construction automatique de taxonomies pour l'aide à la représentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.