LogMap:
Logic-based and Scalable Ontology Matching

Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau

Department of Computer Science, University of Oxford
{ernesto,berg}@cs.ox.ac.uk

Abstract. In this paper, we present LogMap—a highly scalable ontology matching system with 'built-in' reasoning and diagnosis capabilities. To the best of our knowledge, LogMap is the only matching system that can deal with semantically rich ontologies containing tens (and even hundreds) of thousands of classes. In contrast to most existing tools, LogMap also implements algorithms for 'on the fly' unsatisfiability detection and repair. Our experiments with the ontologies NCI, FMA and SNOMED CT confirm that our system can efficiently match even the largest existing bio-medical ontologies. Furthermore, LogMap is able to produce a 'clean' set of output mappings in many cases, in the sense that the ontology obtained by integrating LogMap's output mappings with the input ontologies is consistent and does not contain unsatisfiable classes.

1 Introduction

OWL ontologies are extensively used in biology and medicine. Ontologies such as SNOMED CT, the National Cancer Institute Thesaurus (NCI), and the Foundational Model of Anatomy (FMA) are gradually superseding existing medical classifications and are becoming core platforms for accessing, gathering and sharing bio-medical knowledge and data.

These reference bio-medical ontologies, however, are being developed independently by different groups of experts and, as a result, they use different entity naming schemes in their vocabularies. As a consequence, to integrate and migrate data among applications, it is crucial to first establish correspondences (or *mappings*) between the vocabularies of their respective ontologies.

In the last ten years, the Semantic Web and bio-informatics research communities have extensively investigated the problem of automatically computing mappings between independently developed ontologies, usually referred to as the *ontology matching problem* (see [8] for a comprehensive and up-to-date survey).

The growing number of available techniques and increasingly mature tools, together with substantial human curation effort and complex auditing protocols, has made the generation of mappings between real-world ontologies possible. For example, one of the most comprehensive efforts for integrating bio-medical ontologies through mappings is the UMLS Metathesaurus (UMLS) [2], which integrates more than 100 thesauri and ontologies.

However, despite the impressive state of the art, modern bio-medical ontologies still pose serious challenges to existing ontology matching tools.

Insufficient scalability. Although existing matching tools can efficiently deal with moderately sized ontologies, large-scale bio-medical ontologies such as NCI, FMA or SNOMED CT are still beyond their reach. The largest test ontologies in existing benchmarks (e.g., those in the OAEI initiative) contain around 2000-3000 classes (i.e., with several million possible mappings); however, to the best of our knowledge, no tool has been able to process ontologies with tens or hundreds of thousands of classes (i.e., with several billion possible mappings).

Logical inconsistencies. OWL ontologies have well-defined semantics based on first-order logic, and mappings are commonly represented as OWL class axioms. Hence, the ontology $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$ resulting from the integration of $\mathcal{O}_1$ and $\mathcal{O}_2$ via mappings $\mathcal{M}$ may entail axioms that don't follow from $\mathcal{O}_1$, $\mathcal{O}_2$, or $\mathcal{M}$ alone. Many such entailments correspond to logical inconsistencies due to erroneous mappings in $\mathcal{M}$, or to inherent disagreements between $\mathcal{O}_1$ and $\mathcal{O}_2$. Recent work has shown that even the integration of ontologies via carefully-curated mappings can lead to thousands such inconsistencies [9, 5, 16, 13] (e.g., the integration of FMA-SNOMED via UMLS yields over $6,000$ unsatisfiable classes). Most existing tools are based on lexical matching algorithms, and may also exploit the structure of the ontologies or access external sources such as WordNet; however, these tools disregard the semantics of the input ontologies and are thus unable to detect and repair inconsistencies. Although the first reasoning-based techniques for ontology matching were proposed relatively early on (e.g., S-Match [10]), in practice reasoning is known to aggravate the scalability problem (e.g., no reasoner known to us can classify the integration NCI-SNOMED via UMLS). Despite the technical challenges, there is a growing interest in reasoning techniques for ontology matching. In particular, there has been recent work on 'a-posteriori' mapping debugging [12–15], and a few matching tools (e.g., ASMOV [11], KOSIMap [21], CODI [19, 20]) incorporate techniques for 'on the fly' semantic verification.

In this paper, we present LogMap—a novel ontology matching tool that addresses both of these challenges. LogMap implements highly optimised data structures for lexically and structurally indexing the input ontologies. These structures are used to compute an initial set of *anchor mappings* (i.e., 'almost exact' lexical correspondences) and to assign a confidence value to each of them. The core of LogMap is an iterative process that, starting from the initial anchors, alternates *mapping repair* and *mapping discovery* steps. In order to detect and repair unsatisfiable classes 'on the fly' during the matching process, LogMap implements a sound and highly scalable (but possibly incomplete) ontology reasoner as well as a 'greedy' diagnosis algorithm. New mappings are discovered by iteratively 'exploring' the input ontologies starting from the initial anchor mappings and using the ontologies' *extended class hierarchy*.

To the best of our knowledge, LogMap is the only matching tool that has shown to scale for rich ontologies with tens (even hundreds) of thousands of classes. Furthermore, LogMap is able to produce an 'almost clean' set of output mappings between FMA, SNOMED and NCI; as shown 'a posteriori' using a

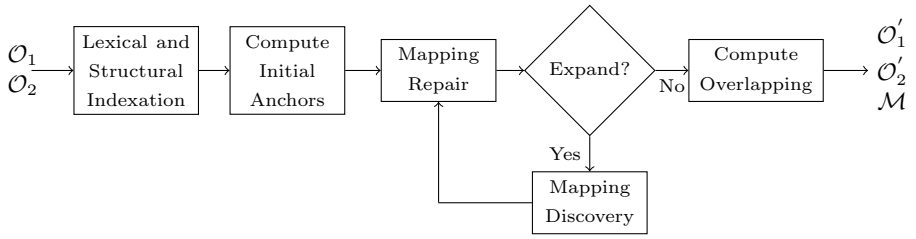


Fig. 1. LogMap in a nutshell.

fully-fledged DL reasoner, LogMap only failed to detect one unsatisfiable class (out of a total of several thousands) when integrating these large-scale ontologies.

2 The Anatomy of LogMap

We next provide an overview of the main steps performed by LogMap, which are schematically represented in Figure 1.

1. **Lexical indexation.** The first step after parsing the input ontologies is their lexical indexation. LogMap indexes the labels of the classes in each ontology as well as their lexical variations, and allows for the possibility of enriching the indexes by using an external lexicon (e.g., WordNet or UMLS-lexicon).
2. **Structural indexation.** LogMap uses an *interval labelling schema* [1, 4, 18] to represent the extended class hierarchy of each input ontology. Each extended hierarchy can be computed using either simple structural heuristics, or an off-the-shelf DL reasoner.
3. **Computation of initial 'anchor mappings'.** LogMap computes an initial set of equivalence *anchor mappings* by intersecting the lexical indexes of each input ontology. These mappings can be considered 'exact' and will later serve as starting point for the further discovery of additional mappings.
4. **Mapping repair and discovery.** The core of LogMap is an iterative process that alternates *repair* and *discovery* steps.
   - In the *repair step*, LogMap uses a sound and highly scalable (but possibly incomplete) reasoning algorithm to detect classes that are unsatisfiable w.r.t. (the merge of) both input ontologies and the mappings computed thus far. Then, each of these undesirable logical consequences is automatically repaired using a 'greedy' diagnosis algorithm.
   - To *discover new mappings*, LogMap maintains two *contexts* (sets of 'semantically related' classes) for each anchor. Contexts for the same anchor are expanded in parallel using the class hierarchies of the input ontologies. New mappings are then computed by matching the classes in the relevant contexts using ISUB [23]—a flexible tool that computes a similarity score for any pair of input strings. This mapping discovery strategy is based on a *principle of locality*: if classes C_1 and C_2 are

Inverted index for NCI labels		**Index for NCI class URIs**	
Entry	***Cls ids***	***Cls id***	***URI***
secretion	49901	49901	NCI:CellularSecretion
cellular,secretion	49901	37975	NCI:Trapezoid
cellular,secrete	49901	62999	NCI:TrapezoidBone
trapezoid	37975,62999	60791	NCI:Smegma
trapezoid,bone	62999		
smegma	60791		
Inverted index for FMA labels		**Index for FMA class URIs**	
Entry	***Cls ids***	***Cls id***	***URI***
secretion	36792	36792	FMA:Secretion
bone,trapezoid	20948,47996	47996	FMA:Bone_of_Trapezoid
trapezoid	20948	20948	FMA:Trapezoid
smegma	60947	60947	FMA:Smegma

Table 1. Fragment of the lexical indexes for NCI and FMA ontologies

correctly mapped, then the classes semantically related to C_1 in $\mathcal{O}_1$ are likely to be mapped to those semantically related to C_2 in $\mathcal{O}_2$.

LogMap continues the iteration of repair and discovery steps until no context is expanded in the discovery step. The output of this process is a set of mappings that are likely to be 'clean'—that is, it will not lead to logical errors when merged with the input ontologies (c.f., evaluation section).

5. **Ontology overlapping estimation.** In addition to the final set of mappings, LogMap computes a fragment of each input ontology, which intuitively represent the 'overlapping' between both ontologies. When manually looking for additional mappings that LogMap might have missed, curators can restrict themselves to these fragments since 'correct' mappings between classes not mentioned in these fragments are likely to be rare.

2.1 Lexical Indexation

LogMap constructs an 'inverted' lexical index (see Table 1) for each input ontology. This type of index, which is commonly used in information retrieval applications, will be exploited by LogMap to efficiently compute an initial set of anchor mappings.

The English name of ontology classes as well as their alternative names (e.g., synonyms) are usually stored in OWL in label annotations. LogMap splits each label of each class in the input ontologies into components; for example, the NCI class 'cellular_secretion' is broken into its component English words 'cellular' and 'secretion'. LogMap allows for the use of an external lexicon (e.g., UMLS lexicon[1] or WordNet) to find both their synonyms and lexical variations; for example, UMLS lexicon indicates that 'secrete' is a lexical variation of 'secretion'.

[1] UMLS Lexicon, unlike WordNet, provides only normalisations and spelling variants.

LogMap groups the component words of each class label and their variations into sets, which will then constitute the key of an inverted index. For example, the inverted index for NCI contains entries for the sets 'cellular, secretion' and 'cellular, secrete'. The range of the index is a numerical ID that LogMap associates to each corresponding class (see Table 1). Thus, in general, an entry in the index can be mapped to several classes (e.g., see 'trapezoid' in Table 1) .

The use of external lexicons to produce a richer index is optional and LogMap allows users to select among well-known lexicons depending on the application.

These indexes can be efficiently computed and bear a low memory overhead. Furthermore, they only need to be computed once for each input ontology.

2.2 Structural Indexation

LogMap exploits the information in the (extended) class hierarchy of the input ontologies in different steps of the matching process. Thus, efficient access to the information in the hierarchies is critical for LogMap's scalability.

The basic hierarchies can be computed by either using structural heuristics, or an off-the-shelf DL reasoner. LogMap bundles HermiT [17] and Condor [22], which are highly optimised for classification. Although DL classification might be computationally expensive, it is performed only once for each ontology.

The class hierarchies computed by LogMap are *extended*—that is, they contain more information than the typical classification output of DL reasoners. In particular, LogMap exploits information about explicit *disjoint classes*, as well as the information in certain complex class axioms (e.g., those stating subsumption between an intersection of named classes and a named class).

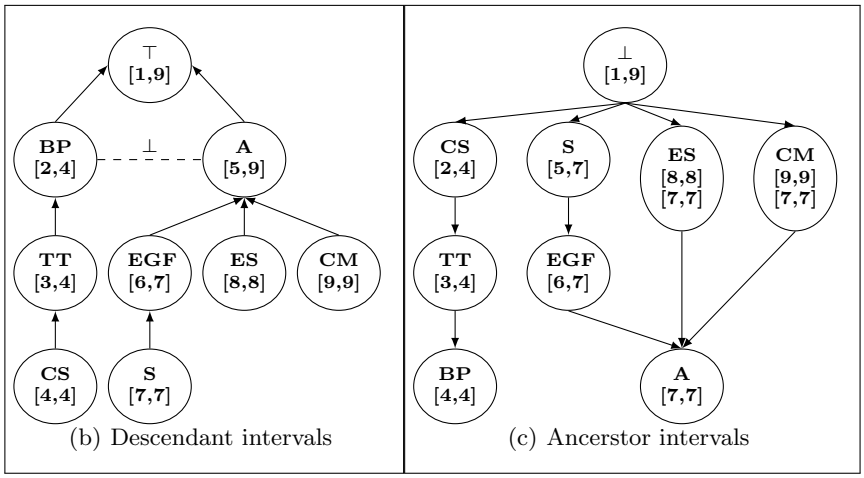
These extended hierarchies are indexed using an interval labelling schema—an optimised data structure for storing DAGs and trees [1]. The use of an interval labelling schema has been shown to significantly reduce the cost of computing typical queries over large class hierarchies [4, 18].

In this context, the ontology hierarchy is treated as two DAGs: the *descendants* DAG representing the descendants relationship, and the *ancestors* DAG, which represents the ancestor relationship. Each named class C in the ontology is represented as a node in each of these DAGs, and is associated with the following information (as in [18]).

- **Descendants preorder number**: $\mathsf{predesc}(C)$ is the order in which C is visited using depth-first traversal of the descendants DAG.
- **Ancestors preorder number**: $\mathsf{preanc}(C)$ is the preorder number of C in the ancestors DAG.
- **Topological order**: deepest associated level within the descendants DAG.
- **Descendants interval:** the information about descendants of C is encoded using the interval [$\mathsf{predesc}(C)$, $\mathsf{maxpredesc}(C)$], where $\mathsf{maxpredesc}(C)$ is the highest preorder number of the children of C in the descendants DAG.
- **Ancestors interval:** the information about ancestors of C is encoded using the interval [$\mathsf{preanc}(C)$,$\mathsf{maxpreanc}(C)$] where maxpreanc(C) is the highest (ancestor) preorder number of the parents of C in the ancestors DAG.

Anatomy $\sqsubseteq \neg$BiologicalProcess
TransmembraneTransport $\sqsubseteq \exists$BP_hasLocation.CellularMembrane
$\exists$BP_hasLocation.$\top \sqsubseteq$ BiologicalProcess
$\top \sqsubseteq \forall$BP_hasLocation.Anatomy
CellularSecretion $\sqsubseteq$ TransmembraneTransport
ExocrineGlandFluid $\sqsubseteq \exists$AS_hasLocation.ExocrineSystem
$\top \sqsubseteq \forall$AS_hasLocation.Anatomy
$\exists$AS_hasLocation.$\top \sqsubseteq$ Anatomy
Smegma $\sqsubseteq$ ExocrineGlandFluid
ExocrineGlandFluid $\sqcap$ ExfoliatedCells $\sqsubseteq$ Smegma

(a) NCI ontology fragment



(b) Descendant intervals (c) Ancerstor intervals

Fig. 2. NCI extended hierarchies. Abbreviations: BP=BiologicalProcess, A=Anatomy, TT=TransmembraneTransport, CM=CellularMembrane, EGF=ExocrineGlandFluid, CS=CellularSecretion, ES=ExocrineSystem, S=Smegma.

Figure 2 shows a fragment of NCI and its labelled (entailed) hierarchy. Disjointness and complex class axioms are represented in a separate structure that also uses integer intervals.

The interval labelling schemas provides LogMap with an interface to efficiently answer queries about taxonomic relationships. For example, the following typical queries over ontology hierarchies only require simple integer operations (please, refer to Figure 2 for the class label abbreviations):

- *'Is* Smegma *a subclass of* Anatomy*?'*: check if predesc(S)=7 is contained in descendants interval [predesc(A), maxpredesc(A)]=[5,9];
- *'Do* Smegma *and* CellularSecretion *have ancestors in common?'*: check if the intersection of ancestors intervals [preanc(S), maxpreanc(S)]=[5,7] and [preanc(CS), maxpreanc(CS)]=[2,4] is non-empty.

Entry	FMA ids	NCI ids	Mappings
secretion	36792	49901	FMA:Secretion ≡ NCI:CellularSecretion
smegma	60947	60791	FMA:Smegma ≡ NCI:Smegma
trapezoid	20948	37975, 62999	FMA:Trapezoid ≡ NCI:Trapezoid FMA:Trapezoid ≡ NCI:TrapezoidBone
trapezoid,bone	20948, 47996	62999	FMA:Trapezoid ≡ NCI:TrapezoidBone FMA:Bone_of_Trapezoid ≡ NCI:TrapezoidBone

Table 2. Fragment of the intersection between the inverted indexes for FMA and NCI

2.3 Computing Anchor Mappings

LogMap computes an initial set of anchor mappings by simply intersecting the inverted indexes of the input ontologies (i.e., by checking whether two lexical entries in the indexes of the input ontologies contain exactly the same strings). Anchor computation is hence extremely efficient. Table 2 shows the result of intersecting the inverted indexes of Table 1, which yields five anchor mappings.

Given an anchor $m = (C_1 \equiv C_2)$, LogMap uses the string matching tool ISUB to match the neighbours of C_1 in the ontology hierarchy of $\mathcal{O}_1$ to the neighbours of C_2 in the hierarchy of $\mathcal{O}_2$. LogMap then assigns a confidence value to m by computing the proportion of matching neighbours weighted by the ISUB similarity values. This technique is based on a principle of locality: if the hierarchy neighbours of the classes in an anchor match with low confidence, then the anchor may be incorrect. For example, LogMap matches classes FMA:Trapezoid and NCI:Trapezoid (see Table 2). However, NCI:Trapezoid is classified as a polygon whereas FMA:Trapezoid is classified as a bone. LogMap assigns a low confidence to such mappings and hence they will be susceptible to be removed during repair.

2.4 Mapping Repair and Discovery

The core of LogMap is an iterative process that alternates *mapping repair* and *mapping discovery* steps. In each iteration, LogMap maintains two structures.

- A working set of *active mappings*, which are mappings that were discovered in the immediately preceding iteration. Mappings found in earlier iterations are *established*, and cannot be eliminated in the repair step. In the first iteration, the active mappings coincide with the set of anchors.
- For each anchor, LogMap maintains two *contexts* (one per input ontology), which can be expanded in different iterations. Each context consists of a set of classes and has a distinguished subset of *active classes*, which is specific to the current iteration. In the first iteration, the contexts for an anchor $C_1 \equiv C_2$ are $\{C_1\}$ and $\{C_2\}$ respectively, which are also the active classes.

Thus, active mappings are the only possible elements of a repair plan, whereas contexts constitute the basis for mapping discovery.

	Propositional FMA ($\mathcal{P}_1$)		Propositional NCI ($\mathcal{P}_2$)
(1)	Smegma → Secretion	(8)	Smegma → ExocrineGlandFluid
(2)	Secretion → PortionBodySusbstance	(9)	ExocrineGlandFluid → Anatomy
(3)	PortionBodySusbstance → AnatomicalEntity	(10)	CellularSecretion → TransmembraneTransport
	Computed mappings ($\mathcal{P}_M$)	(11)	TransmembraneTransport → TransportProcess
(m_4)	FMA:Secretion → NCI:CellularSecretion	(12)	TransportProcess → BiologicalProcess
(m_5)	NCI:CellularSecretion → FMA:Secretion	(13)	Anatomy ∧ BiologicalProcess → false
(m_6)	FMA:Smegma → NCI:Smegma	(14)	ExocrineGlandFluid ∧ ExfolCells → Smegma
(m_7)	NCI:Smegma → FMA:Smegma		

Table 3. Propositional representations of FMA, NCI, and the computed mappings.

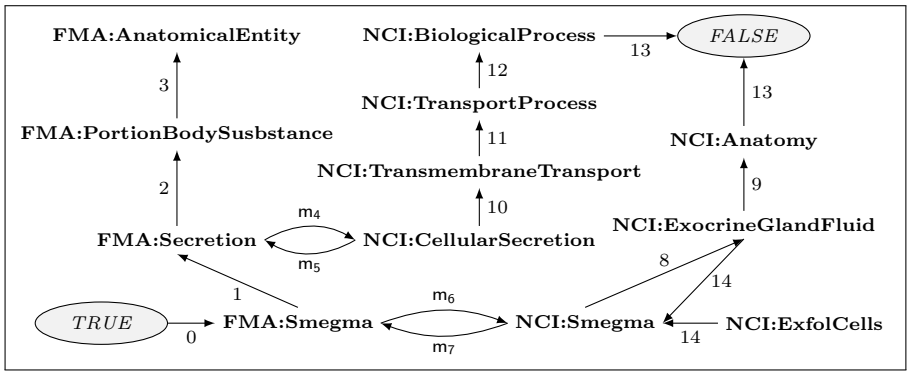


Fig. 3. Graph representation of Horn-clauses in Table 3

.

Mapping Repair LogMap uses a Horn propositional logic representation of the extended hierarchy of each ontology together with all existing mappings (both active and established). As an example, Table 3 shows Horn clauses obtained from the extended hierarchies of FMA and NCI (which have been computed using a DL reasoner), and the anchor mappings computed by LogMap. As shown in the table, LogMap splits each equivalence mapping into two Horn clauses.

The use of a propositional Horn representation for unsatisfiability detection and repair is key to LogMap's scalability since DL reasoners do not scale well with the integration of large ontologies via mappings. The scalability problem is exacerbated by the number of unsatisfiable classes (more than 10,000 found by LogMap when integrating SNOMED and NCI using only anchors) and the large number of additional reasoner calls required for repairing each unsatisfiability.

Unsatisfiability checking LogMap implements the well-known Dowling-Gallier algorithm [7] for propositional Horn satisfiability, and calls the Dowling-Gallier module once (in each repair step) for each class. Our implementation takes as input a class C (represented as a propositional variable) and determines the satisfiability of the propositional theory $\mathcal{P}_C$ consisting of

– the rule ($\mathsf{true} \rightarrow C$);

– the propositional representations $\mathcal{P}_1$ and $\mathcal{P}_2$ (as in Table 3) of the extended hierarchies of the input ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$; and
– the propositional representation $\mathcal{P}_M$ of the mappings computed thus far.

We make the following important observations concerning our encoding of the class satisfiability problem into propositional logic.

– Our encoding is *sound*. If the propositional theory $\mathcal{P}_C$ is unsatisfiable, then the class C is indeed unsatisfiable w.r.t. the DL ontology $\mathcal{O}_1 \cup \mathcal{O}_2 \cup \mathcal{M}$, where $\mathcal{O}_1$ and $\mathcal{O}_2$ are the input ontologies and $\mathcal{M}$ is the set of mappings computed so far by LogMap (represented as DL concept inclusions)
– Due to the properties of the Dowling-Gallier algorithm, our encoding is worst-case linear in the size of $\mathcal{P}_C$. Furthermore, the total number of calls to the Dowling-Gallier module is also linear in the number of classes of $\mathcal{O}_1$ and $\mathcal{O}_2$. As shown in the evaluation section, these favourable computational properties are key to the scalability of LogMap.
– Our encoding is *incomplete*, and hence we might be reporting unsatisfiable classes as satisfiable. Incompleteness is, however, mitigated by the following facts. First, the extended hierarchies of $\mathcal{O}_1$ and $\mathcal{O}_2$ have been computed using a complete reasoner and many consequences that depend on non-propositional reasoning have already been pre-computed. Second, mappings computed by LogMap (and by most ontology matching tools) correspond to Horn rules. For example, as shown in our experiments, LogMap only failed to report one unsatisfiable class for FMA-NCI (from more than 600).

A complete description of the Dowling and Gallier algorithm can be found in [7]. As an example, consider Figure 3, which shows the graph representation of all propositional clauses that are involved in the unsatisfiability of the class Smegma in FMA. Each node represents a propositional variable in Table 3; furthermore, the graph contains a directed edge labelled with a propositional rule r from variable C to variable D if the head of r is D and C occurs in the body of r. Note that there is a path from true to NCI:BiologicalProcess and a path from true to NCI:Anatomy which involve only rules with a single variable in the antecedent; furthermore, the variables NCI:BiologicalProcess and NCI:Anatomy constitute the body of rule (13), whose head is precisely false.

Computing repair plans LogMap computes a *repair* for each unsatisfiable class identified in the input ontologies. Given an unsatisfiable class C and the propositional theory $\mathcal{P}_C$, a *repair* $\mathcal{R}$ of $\mathcal{P}_C$ is a minimal subset of the active mappings in $\mathcal{P}_M$ such that $\mathcal{P}_C \setminus \mathcal{R}$ is satisfiable.

To facilitate computation of repairs, LogMap extends Dowling-Gallier's algorithm to record all *active mappings* ($\mathcal{P}_{act}$) that may be involved in each unsatisfiability. For our example in Figure 3, LogMap records the active mappings $\mathcal{P}_{act} = \{m_4, m_5, m_6, m_7\}$, which may be relevant to the unsatisfiability of FMA:Smegma. This information is used in the subsequent repair process.

To improve scalability, repair computation is based on the 'greedy' algorithm in Table 4. Unsatisfiable classes in each ontology are ordered by their topological

Procedure Repair
Input: *List*: Ordered classes; $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_M$ Horn-propositional theories.
Output: $\mathcal{P}_M$: set of repaired mappings

```
 1: for each C ∈ List do
 2:     P_C := P_1 ∪ P_2 ∪ P_M ∪ {true → C}
 3:     ⟨sat, P_act⟩ := DowlingGallier(P_C)
 4:     if sat = false then
 5:         Repairs := ∅
 6:         repair_size := 1
 7:         repeat
 8:             for each subset R of P_act of size repair_size do
 9:                 sat := DowlingGallier(P_C \ R)
10:                 if sat = true then Repairs := Repairs ∪ {R}
11:             end for
12:             repair_size := repair_size + 1
13:         until |Repairs| > 0
14:         R := element of Repairs with minimum confidence.
15:         P_M := P_M \ R
16:     end if
17: end for
18: return P_M
```

Table 4. Repair in LogMap. A call to DowlingGallier returns a satisfiability value *sat* and, if *sat* = false, it optionally returns the relevant *active mappings* ($\mathcal{P}_{act}$)

level in the hierarchy. Since subclasses of an unsatisfiable class are unsatisfiable, repairing first classes high-up in the hierarchy is a well-known repair strategy.

Given each unsatisfiable class C and the relevant active mappings $\mathcal{P}_{act}$ computed using Dowling-Gallier, the algorithm identifies subsets of $\mathcal{P}_{act}$ of increasing size until a repair is found. Thus, our algorithm is guaranteed to compute all repairs of smallest size. In our example, our algorithm computes repairs $\mathcal{R}_1 = \{m_4\}$ and $\mathcal{R}_2 = \{m_6\}$ consisting of only one mapping. If more than one repair is found, LogMap selects the one with the minimum confidence value.

Finally, each equivalence mapping is split into two propositional rules, which are treated independently for repair purposes. Hence, a repair may include only one such rule, thus 'weakening' the mapping, as in the case of $\mathcal{R}_1$ and $\mathcal{R}_2$.

Mapping Discovery LogMap computes new mappings by first expanding the contexts $\mathbf{C}_1^m$ and $\mathbf{C}_2^m$ for each anchor m, and then (incrementally) matching the classes in $\mathbf{C}_1^m$ to those in $\mathbf{C}_2^m$ using ISUB, as described next.

Context expansion. LogMap only expands contexts that are *open* (i.e., with at least one active class). The expansion of an open context is performed by adding each neighbour (in the corresponding class hierarchy) of an active class in the context. The set of active classes in each context is then reset to the empty set.

Context matching using ISUB. LogMap makes a call to ISUB for each pair of classes $C \in \mathbf{C}_1^m$ and $D \in \mathbf{C}_2^m$, but only if the same call has not been performed

in previous discovery steps (for these or other contexts). Thus, LogMap never calls ISUB twice for the same input classes. We call *relevant* those 'new' lexical correspondences found by ISUB (in the current iteration) with a similarity value exceeding a given *expansion* threshold.

LogMap uses these relevant correspondences to determine the set of active classes of $\mathbf{C}_1^m$ and $\mathbf{C}_2^m$ for the next iteration as well as the set of new mappings.

- The new active classes of $\mathbf{C}_1^m$ and $\mathbf{C}_2^m$ are those that participate in some relevant correspondence.
- The current set of mappings is expanded with those relevant correspondences with similarity value exceeding a *mapping threshold* (which is higher than the expansion threshold). These new mappings will constitute the set of *active mappings* for the next repair step.

The use of ISUB allows LogMap to discover new mappings that, unlike anchors, are not lexically 'exact' (but with similarity higher than the mapping threshold). The number of ISUB tests performed is relatively small: only contexts for the same anchor are matched using ISUB, the same ISUB call is never performed twice, and context growth is limited by the expansion threshold.

2.5 Overlapping Estimation

In addition to the mappings, LogMap also returns two (hopefully small) fragments $\mathcal{O}_1'$ and $\mathcal{O}_2'$ of $\mathcal{O}_1$ and $\mathcal{O}_2$, respectively. Intuitively, $\mathcal{O}_1'$ and $\mathcal{O}_2'$ represent the 'overlapping' between $\mathcal{O}_1$ and $\mathcal{O}_2$, in the sense that each 'correct' mapping not found by LogMap is likely to involve only classes in these fragments. Thus, domain experts can focus only on $\mathcal{O}_1'$ and $\mathcal{O}_2'$ when looking for missing mappings between $\mathcal{O}_1$ and $\mathcal{O}_2$. The computation of $\mathcal{O}_1'$ and $\mathcal{O}_2'$ is performed in two steps.

1. *Computation of 'weak' anchors.* Recall that LogMap computed the initial anchors by checking whether two entries in the inverted index of $\mathcal{O}_1$ and $\mathcal{O}_2$ contained *exactly the same* set of strings (c.f., Section 2.3). For the purpose of overlapping estimation (only), LogMap also computes new anchor mappings that are 'weak' in the sense that the relevant entries in the inverted index are only required to contain *some* common string. Thus, weak anchors represent correspondences between classes that have a common lexical component.
2. *Module extraction.* The sets S_i of classes in $\mathcal{O}_i$ involved in either a weak anchor or a mapping computed by LogMap are then used as 'seed' signatures for module extraction. In particular, $\mathcal{O}_1'$ (resp. $\mathcal{O}_2'$) are computed by extracting a locality-based module [6] for S_1 in $\mathcal{O}_1$ (resp. for S_2 in $\mathcal{O}_2$).

Note that, unlike anchors, 'weak anchors' are not well-suited for mapping computation since they rarely correspond to real mappings, and hence they introduce unmanageable levels of 'noise'. For example, the discovered correspondence NCI:CommonCarotidArteryBranch $\sim$ FMA:BranchOfCommonCochlearArtery is a weak anchor between NCI and FMA because both classes share the terms 'branch', 'common' and 'artery'; however, such correspondence is clearly not a standard mapping since none of the involved classes is subsumed by the other.

	GS Mappings		Repaired Mappings		
Ontologies	**Total**	**Unsat.**	**Total**	$\sqsubseteq$	**Time (s)**
FMA-NCI	3,024	655	(96%) 2,898	78	10.6
FMA-SNOMED	9,072	6,179	(89%) 8,111	1,619	81.4
SNOMED-NCI	19,622	20,944	(93%) 18,322	837	812.4
Mouse-NCI$_{Anat.}$	1,520	0	1,520	-	-

Table 5. Repairing *Gold Standards*. The $\sqsubseteq$ column indicates subsumption mappings. The % of total mappings includes those 'weakened' from equivalence to subsumption.

3 Evaluation

We have implemented LogMap in Java and evaluated it using a standard laptop computer with 4 Gb of RAM.

We have used the following ontologies in our experiments: SNOMED CT Jan. 2009 version (306, 591 classes); NCI version 08.05*d* (66, 724 classes); FMA version 2.0 (78, 989 classes); and NCI Anatomy (3, 304 classes) and Mouse Anatomy (2, 744 classes), both from the OAEI 2010 benchmark [8]. Classification times for these ontologies were the following: 89*s* for SNOMED, 575*s* for NCI, 28*s* for FMA, 1*s* for Mouse Anatomy, and 3*s* for NCI Anatomy.[2] We have performed the following experiments,[3] which we describe in detail in the following sections.

1. *Repair of gold standards*. We have used LogMap's mapping repair module (c.f. Section 2.4) to automatically repair the mappings in two gold standards:
   - The mappings FMA-NCI, FMA-SNOMED and SNOMED-NCI included in UMLS Metathesaurus [2] version 2009AA;[4] and
   - the OAEI 2010 anatomy track gold standard [3].
2. *Matching large ontologies*. We have used LogMap to match the following pairs of ontologies: FMA-NCI, FMA-SNOMED, SNOMED-NCI, and Mouse Anatomy-NCI Anatomy. To the best of our knowledge, no tool has so far matched FMA, NCI and SNOMED; hence, we only compare our results with other tools for the case of Mouse Anatomy-NCI Anatomy.
3. *Overlapping estimation*. We have used LogMap to estimate the overlapping between our test ontologies as described in Section 2.5.

3.1 Repairing Gold Standards

Table 5 summarises our results. We can observe the large number of UMLS mappings between these ontologies (e.g., almost 20, 000 for SNOMED-NCI). Using LogMap we could also detect a large number of unsatisfiable classes (ranging from 655 for FMA-NCI to 20, 944 for SNOMED-NCI), which could be repaired efficiently (times range from 10.6*s* for FMA-NCI to 812.4*s* for SNOMED-NCI).

[2] We used ConDOR [22] to classify SNOMED, and HermiT [17] for the others.
[3] Output resources available in: `http://www.cs.ox.ac.uk/isg/projects/LogMap/`
[4] The mappings are extracted from the UMLS distribution files (see [13] for details).

	Found Mapp.		Output Mapp.		Time (s)	
Ontologies	**Total**	**Unsat.**	**Total**	$\sqsubseteq$	**Anchors**	**Total**
FMA-NCI	3,185	597	(94%) 3,000	43	28.3	69.8
FMA-SNOMED	2,068	570	(99%) 2,059	32	35.6	92.2
SNOMED-NCI	14,250	10,452	(95%) 13,562	1,540	528.6	1370.0
Mouse-NCI$_{Anat}$	1,369	32	(99%) 1,367	3	1.8	15.7

Table 6. Mappings computed by LogMap

Finally, the repair process was not aggressive, as it resulted in the deletion of a small number of mappings;[5] for example, in the case of NCI and FMA LogMap preserved 96% of the original mappings, and also managed to 'weaken' 78 equivalence mappings into subsumption mappings (instead of deleting them).

We have used the reasoners HermiT and ConDOR to classify the merge of the ontologies and the repaired mappings, thus verifying the results of the repair. For FMA-NCI, we found one unsatisfiable class that was not detected by LogMap's (incomplete) reasoning algorithm. Unsatisfiability was due to a complex interaction of three 'exact' lexical mappings with axioms in NCI and FMA involving existential and universal restrictions. For FMA-SNOMED and SNOMED-NCI we could not classify the merged ontologies, so we extracted a module [6] of the mapped classes in each ontology. For FMA-SNOMED we could classify the merge of the corresponding modules and found no unsatisfiable classes. For SNOMED-NCI no reasoner could classify the merge of the modules.

In the case of the Mouse Anatomy and NCI Anatomy ontologies from OEAI, we found no unsatisfiable class using both LogMap and a DL reasoner.

3.2 Matching Large Ontologies

Table 6 summarises the results obtained when matching our test ontologies using LogMap for a default expansion threshold of 0.70 and mapping threshold of 0.95.

The second and third columns in Table 6 indicate the total number of mappings found by LogMap (in all repair-discovery iterations), and the total number of detected unsatisfiable classes, respectively. The fourth and fifth columns provide the total number of output mappings (excluding those discarded during repair) and shows how many of those mappings were 'weakened' from equivalence to simple subsumption during the repair process. We can observe that, despite the large number of unsatisfiable classes, the repair process was not aggressive and more than 94% (in the worst case) of all discovered mappings were returned as output. Finally, the last two columns show the times for anchor computation and repair, and the total matching time.[6]

Total matching time (including anchor computation and repair-discovery iterations) was less than two minutes for FMA-NCI and FMA-SNOMED. The

[5] The repair process in our prior work was much more aggressive [13]; for example, 63% of UMLS for SNOMED-NCI were deleted.

[6] Excluding only indexation time, which is negligible.

	Found Mappings			Output Mappings		
Ontologies	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
FMA-NCI	0.767	0.843	0.803	0.811	0.840	0.825
FMA-SNOMED	0.767	0.195	0.312	0.771	0.195	0.312
SNOMED-NCI	0.753	0.585	0.659	0.786	0.582	0,668
Mouse-NCI$_{Anat}$	0.917	0.826	0.870	0.918	0.826	0.870

Table 7. Precision and recall w.r.t. Gold Standard.

	GS ISUB $\geq$ 0.95		GS ISUB $\geq$ 0.80		GS ISUB $\geq$ 0.50	
Ontologies	**% Mapp.**	**Recall**	**% Mapp.**	**Recall**	**% Mapp.**	**Recall**
FMA-NCI	88%	0.96	93%	0.90	97%	0.87
FMA-SNOMED	21%	0.95	64%	0.30	92%	0.21
SNOMED-NCI	62%	0.94	75%	0.77	89%	0.65
Mouse-NCI$_{Anat}$	75%	0.99	87%	0.95	95%	0.88

Table 8. Missed mappings by LogMap with respect to repaired gold standard

slowest result was obtained for SNOMED-NCI (20 minutes) since repair was costly due to the huge number of unsatisfiable classes. We could only compare performance with other tools for Mouse-NCI$_{Anat}$ (the largest ontology benchmark in the OAEI). LogMap matched these ontologies in 15.7 seconds, whereas the top three tools in the 2009 campaign (no official times in 2010) required 19, 23 and 10 minutes, respectively; furthermore, the CODI tool, which uses sophisticated logic-based techniques to reduce unsatisfiability, reported times between 60 to 157 minutes in the 2010 OAEI [20].

Table 7 shows precision and recall values w.r.t. our Gold Standards (the 'clean' UMLS-Mappings from our previous experiment and the mappings in the anatomy track of the OAEI 2010 benchmark). The left-hand-side of the table shows precision/recall values for the set of all mappings found by LogMap (by disabling the repair module), whereas the right-hand-side shows precision/recall for the actual set of output mappings. Our results can be summarised as follows:

- Although the main benefit of repair is to prevent logical errors, the table shows that repair also increases precision without harming recall.
- In the case of Mouse-NCI$_{Anat}$ we obtained an F-score in line with the best systems in the 2010 OAEI competition [8].
- Results for FMA-NCI were very positive, with both precision and recall exceeding 0.8. Although precision was also high for SNOMED-NCI and FMA-SNOMED, recall values were much lower, especially for FMA-SNOMED.

We have analysed the reason for the low recall values for FMA-SNOMED and SNOMED-NCI. Our hypothesis was that SNOMED is 'lexically incompatible' with FMA and NCI since it uses very different naming conventions. Results in Table 8 support this hypothesis. Table 8 shows, on the one hand, the percentage of gold standard mappings with an ISUB similarity exceeding a given threshold and, on the other hand, the recall values for LogMap w.r.t. such mappings only.

Ontologies	Overlapping for $\mathcal{O}_1$			Overlapping for $\mathcal{O}_2$		
$\mathcal{O}_1$-$\mathcal{O}_2$	$\mathcal{O}'_1$	% $\mathcal{O}_1$	**Recall**	$\mathcal{O}'_2$	% $\mathcal{O}_2$	**Recall**
FMA-NCI	6,512	8%	0.95	12,867	19%	0.97
FMA-SNOMED	20,278	26%	0.92	50,656	17%	0.94
SNOMED-NCI	70,705	23%	0.86	33,829	51%	0.96
Mouse-NCI$_{Anat}$	1,864	68%	0.93	1,894	57%	0.93

Table 9. Overlapping computed by LogMap

Note that LogMap could find in all cases more than 94% of the gold standard mappings having ISUB similarity above 0.95. However, only 21% of the gold standard FMA-SNOMED mappings exceeded this value (in contrast to 88% between FMA and NCI), showing that these ontologies use very different naming conventions. To achieve a high recall for FMA-SNOMED mappings, LogMap would need to use a mapping threshold of 0.5, which would introduce an unmanageable amount of 'noisy' mappings, thus damaging both precision and scalability.

3.3 Overlapping Estimation

Our results concerning overlapping are summarised in Table 9, where $\mathcal{O}'_1$ and $\mathcal{O}'_2$ are the fragments of the input ontologies computed by LogMap.

We can see that the output fragments are relatively small (e.g., only 8% of FMA and 19% of NCI for FMA-NCI and only 26% of FMA and 17% of SNOMED for FMA-SNOMED). Our results also confirm the hypothesis that 'correct' mappings involving an entity outside these fragments are rare. As shown in the table, a minimum of 86% and a maximum of 97% of Gold Standard UMLS mappings involve *only* classes in the computed fragments. Thus, these results confirm our hypothesis even for FMA-SNOMED and SNOMED-NCI, where LogMap could only compute a relatively small fraction of the Gold Standard mappings.

4 Conclusion and Future Work

In this paper, we have presented LogMap—a highly scalable ontology matching tool with built-in reasoning and diagnosis capabilities. LogMap's features and scalability behaviour make it well-suited for matching large-scale ontologies. LogMap, however, is still an early-stage prototype and there is plenty of room for improvement. We are currently working on further optimisations, and in the near future we are planning to integrate LogMap with a Protege-based front-end, such as the one implemented in our tool ContentMap [12].

Acknowledgements

We would like to acknowledge the funding support of the Royal Society and the EPSRC project *LogMap*, and also thank V. Nebot and R. Berlanga for their support in our first experiments with structural indexation.

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. SIGMOD Rec. 18, 253–262 (1989)
2. Bodenreider, O.: The Unified Medical Language System (UMLS): integrating biomedical terminology. Nucleic acids research 32 (2004)
3. Bodenreider, O., Hayamizu, T.F., et al.: Of mice and men: Aligning mouse and human anatomies. In: AMIA Annu Symp Proc. pp. 61–65 (2005)
4. Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: On labeling schemes for the Semantic Web. In: Proc. of WWW. pp. 544–555. ACM (2003)
5. Cimino, J.J., Min, H., Perl, Y.: Consistency across the hierarchies of the UMLS semantic network and metathesaurus. J of Biomedical Informatics 36(6) (2003)
6. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: extracting modules from ontologies. In: Proc. of WWW. pp. 717–726 (2007)
7. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. J. Log. Program. pp. 267–284 (1984)
8. Euzenat, J., Meilicke, C., Stuckenschmidt, H., Shvaiko, P., Trojahn, C.: Ontology Alignment Evaluation Initiative: six years of experience. J Data Semantics (2011)
9. Geller, J., Perl, Y., Halper, M., Cornet, R.: Special issue on auditing of terminologies. Journal of Biomedical Informatics 42(3), 407–411 (2009)
10. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: an algorithm and an implementation of semantic matching. In: European Semantic Web Symposium (2004)
11. Jean-Mary, Y.R., Shironoshita, E.P., Kabuka, M.R.: Ontology matching with semantic verification. J of Web Semantics 7(3), 235–251 (2009)
12. Jimenez-Ruiz, E., Cuenca Grau, B., Horrocks, I., Berlanga, R.: Ontology integration using mappings: Towards getting the right logical consequences. In: Proc. of European Semantic Web Conference (ESWC). pp. 173–187 (2009)
13. Jiménez-Ruiz, E., Cuenca Grau, B., Horrocks, I., Berlanga, R.: Logic-based assessment of the compatibility of UMLS ontology sources. J Biomed. Sem. 2 (2011)
14. Meilicke, C., Stuckenschmidt, H.: An efficient method for computing alignment diagnoses. In: Proc. of Web Reasoning and Rule Systems, RR. pp. 182–196 (2009)
15. Meilicke, C., Stuckenschmidt, H., Tamilin, A.: Reasoning Support for Mapping Revision. J Logic Computation 19(5), 807–829 (2009)
16. Morrey, C.P., Geller, J., Halper, M., Perl, Y.: The Neighborhood Auditing Tool: A hybrid interface for auditing the UMLS. J of Biomedical Informatics 42(3) (2009)
17. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research 36, 165–228 (2009)
18. Nebot, V., Berlanga, R.: Efficient retrieval of ontology fragments using an interval labeling scheme. Inf. Sci. 179(24), 4151–4173 (2009)
19. Niepert, M., Meilicke, C., Stuckenschmidt, H.: A probabilistic-logical framework for ontology matching. In: Proc. of AAAI (2010)
20. Noessner, J., Niepert, M.: CODI: Combinatorial optimization for data integration results for OAEI 2010. In: Proc. of OM Workshop (2010)
21. Reul, Q., Pan, J.Z.: KOSIMap: Use of description logic reasoning to align heterogeneous ontologies. In: Proc. of DL Workshop (2010)
22. Simancik, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: IJCAI (2011)
23. Stoilos, G., Stamou, G.B., Kollias, S.D.: A string metric for ontology alignment. In: Proc. of the International Semantic Web Conference (ISWC). pp. 624–637 (2005)