Int J Digit Libr (2012) 12:179–193
DOI 10.1007/s00799-012-0083-2

Computing minimal mappings between lightweight ontologies

Fausto Giunchiglia · Vincenzo Maltese · Aliaksandr Autayeu

Published online: 11 February 2012

Abstract As a valid solution to the semantic heterogeneity problem, many matching solutions have been proposed. Given two lightweight ontologies, we compute the minimal mapping, namely the subset of all possible correspondences, that we call mapping elements, between them such that (i) all the others can be computed from them in time linear in the size of the input ontologies and (ii) none of them can be dropped without losing property (i). We provide a formal definition of minimal mappings and define a time efficient computation algorithm which minimizes the number of comparisons between the nodes of the two input ontologies. The experimental results show a substantial improvement both in the computation time and in the number of mapping elements which need to be handled, for instance for validation, navigation, and search.

Keywords Interoperability · Ontology matching · Lightweight ontologies · Minimal mappings

This article is an integrated and extended version of two papers. The first was entitled "Computing minimal mapping" and was presented at the 4th Workshop on Ontology Matching [15]; the second was entitled "Save up to 99% of your time in mapping validation" and was presented at the 9th International ODBASE Conference [22].

F. Giunchiglia · V. Maltese (✉) · A. Autayeu
Dipartimento di Ingegneria e Scienza dell'Informazione (DISI), Università di Trento, Trento, Italy
e-mail: maltese@disi.unitn.it

F. Giunchiglia
e-mail: fausto@disi.unitn.it

A. Autayeu
e-mail: autayeu@disi.unitn.it

1 Introduction

Given any two graph-like structures, e.g., database and XML schemas, classifications, thesauri, and ontologies, matching is usually identified as the problem of finding those nodes in the two structures which semantically correspond to one another. Any such pair of nodes, along with the semantic relationship holding between the two, is what we call a *mapping element*. In the last few years, a lot of study has been done on this topic both in the digital libraries [35,19,23,7,18,26,33,27,20] and the computer science [28,29,21,12,5,11,31] communities.

We concentrate on lightweight ontologies (or formal classifications), as formally defined in [10,14]. This must not be seen as a limitation. There are plenty of schemas in the world which can be translated, with almost no loss of information, into lightweight ontologies. For instance, thesauri, library classifications, file systems, email folder structures, web directories, business catalogs, and so on. Lightweight ontologies are well defined and pervasive. We focus on the problem of computing the *minimal mapping*, namely the subset of all possible *mapping elements* such that (i) all the others can be computed from them in time linear in the size of the input graphs and (ii) none of them can be dropped without losing property (i). The minimal mapping is the set of maximum size with no redundant elements. By concentrating on lightweight ontologies, we are able to identify specific redundancy patterns that are at the basis of what we define to be a *minimal mapping*. Such patterns are peculiar of lightweight ontologies and not of all ontologies in general. The main advantage of minimal mappings is that they are the minimal amount of information that needs to be dealt with. Notice that this is a rather important feature as the number of possible mapping elements can grow up to $n * m$ with n and m being the size of the two input ontologies. Minimal mappings provide clear usability advantages. Many systems and corresponding interfaces, mostly graphical, have been provided for the management of mappings but all of them hardly scale with the increasing number of nodes, and the resulting visualizations are rather messy [29]. Furthermore, the maintenance of smaller sets makes the study of the user much easier, faster, and less error-prone [25].

Our main contributions are (a) a formal definition of *minimal* and, dually, *redundant mappings*, (b) evidence of the fact that the minimal mapping always exists and it is unique, and (c) an algorithm to compute it. This algorithm has the following main features:

1. It can be proved to be correct and complete, in the sense that it always computes the minimal mapping;
2. It is very efficient as it minimizes the number of calls to the node matching function, namely to the function which computes the relation between two nodes. Notice that node matching in the general case amounts to logical reasoning (i.e., SAT reasoning) [12], and it may require exponential time;
3. To compute the set of all correspondences between the two ontologies, it computes the *mapping of maximum size* (including the maximum number of redundant elements). This is done by maximally exploiting the information codified in the graph of the lightweight ontologies in input. This, in turn, helps to avoid missing mapping elements due to pitfalls in the node matching functions, such as missing background knowledge [11].

As far as we know very little study has been done on the issue of computing minimal mappings. In general, the computation of minimal mappings can be seen as a specific instance of the mapping inference problem [21]. Closer to our study, in [31,24,25] the authors use distributed description logics (DDL) [4] to represent and reason about existing ontology mappings. They introduce a few debugging heuristics which remove mapping elements which are redundant or generate inconsistencies in a given set [24]. The main problem of this approach, as also recognized by the authors, is the complexity of DDL reasoning [25]. In our approach, by concentrating on lightweight ontologies, instead of pruning redundant elements, we directly compute the minimal set. Among other things, our approach allows minimizing the number of calls to the node matching functions.

The rest of the article is organized as follows. Section 2 provides a motivating example and shows how we convert classifications into lightweight ontologies. Section 3 provides the definition for redundant and minimal mappings, and it shows that the minimal set always exists and it is unique. Section 4 describes the algorithm, while Sect. 5 evaluates it. Finally, Sect. 6 summarizes and concludes the article.

2 Converting classifications into lightweight ontologies

Classifications are perhaps the most natural tool humans use to organize information content. Information items are hierarchically arranged under topic nodes moving from general ones to more specific ones as long as we go deeper in the hierarchy. This attitude is well known in knowledge organization as the principle of organizing from the general to the specific [19], called synthetically the *get-specific principle* in [10,14]. Consider the two fragments of classifications depicted in Fig. 1. They are designed to arrange more or less the same content, but from different perspectives. The second is a fragment taken from the Yahoo web directory[1] (category Computers and Internet).

Following the approach described in [10] and exploiting dedicated NLP techniques tuned to short phrases (for instance, as described in [34]), classifications can be converted, exactly or with a certain degree of approximation, into their formal alter-ego, namely into lightweight ontologies. Lightweight ontologies are acyclic graph structures where each natural language node label is translated into a propositional description logic (DL) formula codifying the meaning of the node. Note that the formula associated to each node contains the formula of the node above to capture the fact that the meaning of each node is contextualized by the meaning of its ancestor nodes. As a consequence, the backbone structure of the resulting lightweight ontologies is represented by subsumption relations between nodes. The resulting formulas are reported in Fig. 2.

Here, each string denotes a concept (such as journals#1) and the number at the end of the string denotes a specific concept constructed from a WordNet synset. Figure 2 also reports the resulting mapping elements. We assume that each mapping element is associated with one of the following semantic relations: disjointness ($\perp$), equivalence ($\equiv$), more specific ($\sqsubseteq$), and less specific ($\sqsupseteq$), as computed for instance by semantic matching [12]. Notice that not all the mapping elements have the same semantic valence. For instance, $B \sqsubseteq D$ is a trivial logical consequence of $B \sqsubseteq E$ and $E \sqsubseteq D$, and similarly for $C \sqsubseteq F$ and $C \equiv G$. We represent the elements in the minimal mapping using solid lines and redundant elements using dashed lines. M' is the set of maximum size (including the maximum number of redundant elements) while M is the minimal set. The problem we address in the following is how to compute the minimal set in the most efficient way.

3 Redundant and minimal mappings

Adapting the definition in [10], we define a lightweight ontology as follows:

[1] http://dir.yahoo.com/.

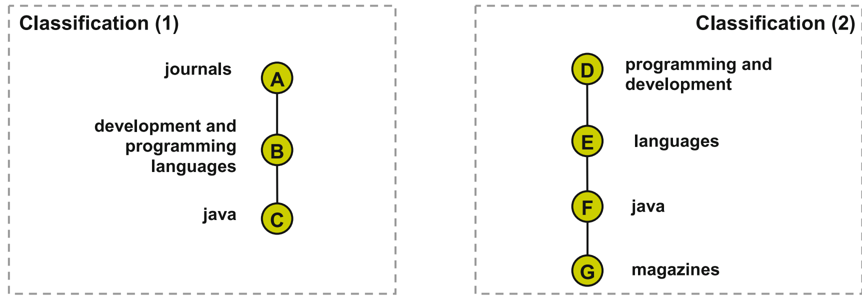


Fig. 1 Two classifications

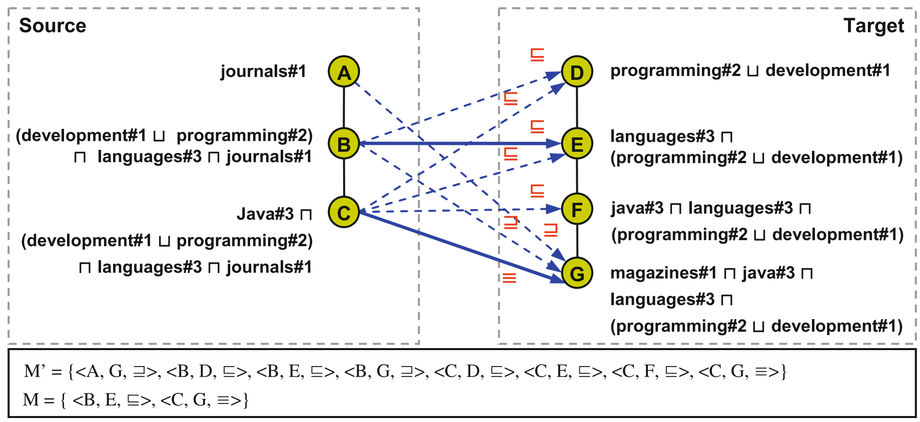


Fig. 2 The minimal and redundant mapping between two lightweight ontologies

Definition 1 *(Lightweight ontology)* A lightweight ontology O is a rooted tree $\langle N,E,L^F\rangle$ where:

(a) N is a finite set of nodes;
(b) E is a set of edges on N;
(c) L^F is a finite set of labels expressed in a Propositional DL language such that for any node $n_i \in N$, there is one and only one label $l_i^F \in L^F$;
(d) $l_{i+1}{}^F \sqsubseteq l_i^F$ with n_i being the parent of n_{i+1}.

The superscript F is used to emphasize that labels are in a formal language. Figure 2 provides an example of (a fragment of) two lightweight ontologies.

We then define mapping elements as follows:

Definition 2 *(Mapping element)* Given two lightweight ontologies O_1 and O_2, a mapping element m between them is a triple $\langle n_1, n_2, R\rangle$, where:

(a) $n_1 \in N_1$ is a node in O_1, called the source node;
(b) $n_2 \in N_2$ is a node in O_2, called the target node;
(c) $R \in \{\bot, \equiv, \sqsubseteq, \sqsupseteq\}$ is the strongest semantic relation holding between n_1 and n_2.

The partial order is such that disjointness is stronger than equivalence which, in turn, is stronger than subsumption (in both directions), and such that the two subsumption symbols are unordered. This is to return subsumption only when equivalence does not hold or one of the two nodes being inconsistent (this latter case generating at the same time both a disjointness and a subsumption relation), and similarly for the order between disjointness and equivalence. Notice that under this ordering there can be at most one mapping element between two nodes.

The next step is to define the notion of redundancy. The key idea is that given a mapping element $\langle n_1, n_2, R\rangle$, a new mapping element $\langle n'_1, n'_2, R'\rangle$ is redundant with respect to the first if the existence of the second can be asserted simply by looking at the relative positions of n_1 with n'_1, and n_2 with n'_2. In algorithmic terms, this means that the second can be computed without running the time expensive node matching functions. We have identified four basic redundancy patterns as follows:

In Fig. 3, straight solid arrows represent minimal mapping elements, dashed arrows represent redundant mapping elements, and curves represent redundancy propagation. Let us discuss the rationale for each of the patterns:

- **Pattern (1)**: each mapping element $\langle C, D, \sqsubseteq\rangle$ is redundant w.r.t. $\langle A, B, \sqsubseteq\rangle$. In fact, C is more specific than A which is more specific than B which is more specific than D. As a consequence, by transitivity C is more specific than D.
- **Pattern (2)**: dual argument as in pattern (1).
- **Pattern (3)**: each mapping element $\langle C, D, \bot\rangle$ is redundant w.r.t. $\langle A, B, \bot\rangle$. In fact, we know that A and B are

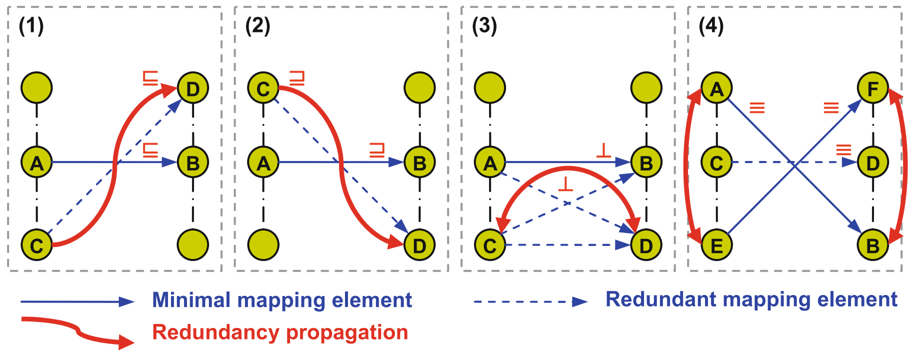


Fig. 3 Redundancy detection patterns

disjoint, that C is more specific than A and that D is more specific than B. This implies that C and D are also disjoint.

- **Pattern (4)**: Pattern 4 is the combination of patterns (1) and (2).

In other words, the patterns are the way to capture logical inference from structural information, namely just by looking at the position of the nodes in the two trees. Note that they capture the logical inference w.r.t. one mapping element only. As we will show, this in turn allows computing the redundant elements in linear time (w.r.t. the size of the two ontologies) from the ones in the minimal set.

Notice that patterns (1) and (2) are still valid in case we substitute subsumption with equivalence. However, in this case we cannot exclude the possibility that a stronger relation holds. A trivial example of where this is not the case is provided in Fig. 4a.

On the basis of the patterns and the considerations above, we can define redundant elements as follows. Here path(n) is the path from the root to the node n.

Definition 3 *(Redundant mapping element)* Given two lightweight ontologies O_1 and O_2, a mapping M and a mapping element $m' \in M$ with $m' = \langle C, D, R' \rangle$ between them, we say that m' is redundant in M iff one of the following holds:

(1) If R' is $\sqsubseteq$, $\exists m \in M$ with $m = \langle A, B, R \rangle$ and $m \neq m'$ such that $R \in \{\sqsubseteq, \equiv\}$, $A \in path(C)$ and $D \in path(B)$;
(2) If R' is $\sqsupseteq$, $\exists m \in M$ with $m = \langle A, B, R \rangle$ and $m \neq m'$ such that $R \in \{\sqsupseteq, \equiv\}$, $C \in path(A)$ and $B \in path(D)$;
(3) If R' is $\perp$, $\exists m \in M$ with $m = \langle A, B, \perp \rangle$ and $m \neq m'$ such that $A \in path(C)$ and $B \in path(D)$;
(4) If R' is $\equiv$, conditions (1) and (2) must be satisfied.

See how Definition 3 maps to the four patterns in Fig. 3. Figure 2 provides examples of redundant elements. Definition 3 can be proved to capture all and only the cases of redundancy.

Theorem 1 (Redundancy, soundness and completeness) *Given a mapping M between two lightweight ontologies O_1 and O_2, a mapping element $m' \in M$ is logically redundant w.r.t. another mapping element $m \in M$ if and only if it satisfies one of the conditions of Definition 3.*

The soundness argument is the rationale described for the patterns above. Completeness can be shown by constructing the counterargument that we cannot have redundancy in the remaining cases. We can proceed by enumeration, negating each of the patterns, encoded one by one in the conditions appearing in the Definition 3. The complete proof is given in the Appendix. Figure 4b provides an example of non redundancy which is based on pattern (1). It tells us that the existence of a correspondence between two nodes does not necessarily propagate to the two nodes below. For example we cannot derive that Canine $\sqsubseteq$ Dog from the set of axioms {Canine $\sqsubseteq$ Mammal, Mammal $\sqsubseteq$ Animal, Dog $\sqsubseteq$ Animal}, and it would be wrong to do so.

Using the notion of redundancy, we formalize the minimal mapping as follows:

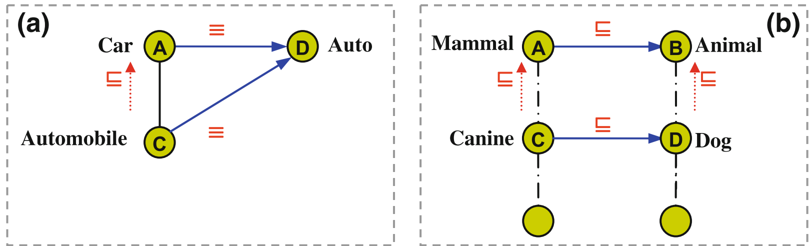
Definition 4 *(Minimal mapping)* Given two lightweight ontologies O_1 and O_2, we say that a mapping M between them is minimal iff:

a) $\nexists m \in M$ such that m is redundant (minimality condition);
b) $\nexists M' \supset M$ satisfying condition (a) above (maximality condition).

A mapping element is minimal if it belongs to the minimal mapping.

Note that conditions (a) and (b) ensure that the minimal set is the set of maximum size with no redundant elements. As an example, the set M in Fig. 2 is minimal. Comparing this mapping with M' we can observe that all elements in the set M'–M are redundant and that, therefore, there are no other supersets of M with the same properties. In effect, $\langle A, G, \sqsupseteq \rangle$ and $\langle B, G, \sqsupseteq \rangle$ are redundant w.r.t. $\langle C, G, \equiv \rangle$ for pattern (2); $\langle C, D, \sqsubseteq \rangle$, $\langle C, E, \sqsubseteq \rangle$ and $\langle C, F, \sqsubseteq \rangle$ are redundant w.r.t. $\langle C, G, \equiv \rangle$ for pattern (1); $\langle B, D, \sqsubseteq \rangle$ is redundant w.r.t. $\langle B, E, \sqsubseteq \rangle$ for pattern (1). Note that M contains far less mapping elements w.r.t. M'.

Fig. 4 Examples of non-redundant mapping elements



As last observation, for any two given lightweight ontologies, the minimal mapping always exists and it is unique. In fact, Definition 3 imposes a strict partial order over mapping elements. In other words, given two elements m and m′, m′⟨m if and only if m′ is redundant w.r.t. m. Under the strict partial order above, the minimal mapping is the set of all the maximal elements of the partially ordered set.

Keeping in mind the patterns in Fig. 3, the minimal set can be efficiently computed using the following key intuitions:

1. Equivalence can be "opened" into two subsumption mapping elements of opposite direction;
2. Taking any two paths in the two ontologies, a minimal subsumption mapping element (in both directions of subsumption) is an element with the highest node in one path whose formula is subsumed by the formula of the lowest node in the other path;
3. Taking any two paths in the two ontologies, a minimal disjointness mapping element is the one with the highest nodes in both paths such that their formulas satisfy disjointness.

4 Computing minimal and redundant mappings

The patterns described in the previous section allow us not only to identify minimal and redundant mapping elements, but they also suggest how to significantly reduce the amount of calls to the node matchers. By looking for instance at pattern (2) in Fig. 3, given a mapping element m = ⟨A, B, ⊒⟩ we know in advance that it is not necessary to compute the semantic relation holding between A and any descendant C in the sub-tree of B as we know in advance that it is ⊒. At the top level the algorithm is organized as follows:

- **Step 1: computing the minimal mapping modulo equivalence:** compute the set of disjointness and subsumption mapping elements which are *minimal modulo equivalence*. By this we mean that they are minimal modulo collapsing, whenever possible, two subsumption relations of opposite direction into a single equivalence mapping element;
- **Step 2: computing the minimal mapping:** eliminate the redundant subsumption mapping elements. In particular, collapse all the pairs of subsumption elements (of opposite direction) between the same two nodes into a single equivalence element. This will result into the *minimal mapping*;
- **Step 3: computing the mapping of maximum size:** compute the mapping of maximum size (including minimal and redundant mapping elements). During this step the existence of a (redundant) element is computed as the result of the propagation of the elements in the minimal mapping. Notice that redundant equivalence mapping elements can be computed due to the propagation of minimal equivalence elements or of two minimal subsumption elements of opposite direction. However, it can be easily proved that in the latter case they correspond to two partially redundant equivalence elements, where a partially redundant equivalence element is an equivalence element where one direction is a redundant subsumption mapping element while the other is not.

The first two steps are performed at matching time, while the third is activated whenever the user wants to exploit the pre-computed mapping elements, e.g., for their visualization. The following three subsections analyze the three steps above in detail.

4.1 Step 1: Computing the minimal mapping modulo equivalence

The minimal mapping is computed by a function **TreeMatch** whose pseudo-code is provided in Fig. 5. M is the minimal set, while T1 and T2 are the input lightweight ontologies. **TreeMatch** is crucially dependent on the two node matching functions **NodeDisjoint** (Fig. 6) and **NodeSubsumedBy** (Fig. 7) which take two nodes n1 and n2 and return a positive answer in case of disjointness and subsumption, or a negative answer if it is not the case or they are not able to establish it. Note that these two functions hide the heaviest computational costs. In particular, their computation time is exponential when the relation holds and exponential in the worst case, but possibly much faster, when the relation does not hold. The main motivation for this is that the node matching

Fig. 5 Pseudo-code for the tree matching function

```
node: struct of {cnode: wff; children: node[];}
T1,T2: tree of (node);
relation in {⊑, ⊒, ≡, ⊥};
element: struct of {source: node; target: node; rel: relation;};
M: list of (element);
boolean direction;
function TreeMatch(tree T1, tree T2)
 {TreeDisjoint(root(T1),root(T2));
  direction := true;
  TreeSubsumedBy(root(T1),root(T2));
  direction := false;
  TreeSubsumedBy(root(T2),root(T1));
  TreeEquiv();
 };
```

Fig. 6 Pseudo-code for the **TreeDisjoint** function

```
function TreeDisjoint(node n1, node n2)
 {c1: node;
  NodeTreeDisjoint(n1, n2);
  foreach c1 in GetChildren(n1) do TreeDisjoint(c1,n2);
 };
function NodeTreeDisjoint(node n1, node n2)
 {n,c2: node;
  foreach n in Path(Parent(n1)) do if (<n,n2,⊥> ∈ M) then return;
  if (NodeDisjoint(n1, n2)) then
    {AddMappingElement(<n1,n2,⊥>);
     return;
    };
  foreach c2 in GetChildren(n2) do NodeTreeDisjoint(n1,c2);
 };
function boolean NodeDisjoint(node n1, node n2)
 {if (Unsatisfiable(mkConjunction(n1.cnode,n2.cnode))) then
    return true;
  else return false; };
```

Fig. 7 Pseudo-code for the **TreeSubsumedBy** function

```
function boolean TreeSubsumedBy(node n1, node n2)
 {c1,c2: node; LastNodeFound: boolean;
  if (<n1,n2,⊥> ∈ M) then return false;
  if (!NodeSubsumedBy(n1, n2)) then
    foreach c1 in GetChildren(n1) do TreeSubsumedBy(c1,n2);
  else
    {LastNodeFound := false;
     foreach c2 in GetChildren(n2) do
       if (TreeSubsumedBy(n1,c2)) then LastNodeFound := true;
     if (!LastNodeFound) then AddSubsumptionMappingElement(n1,n2);
     return true;
    };
  return false;
 };
function boolean NodeSubsumedBy(node n1, node n2)
 {if (Unsatisfiable(mkConjunction(n1.cnode,negate(n2.cnode)))) then
    return true;
  else return false; };
function AddSubsumptionMappingElement(node n1, node n2)
 {if (direction) then AddMappingElement(<n1,n2,⊑>);
  else AddMappingElement(<n2,n1,⊒>); };
```

problem, in the general case, should be translated into disjointness or subsumption problem in propositional DL (see [12] for a detailed description).

The goal, therefore, is to compute the minimal mapping by minimizing the calls to the node matching functions and, in particular minimizing the calls where the relation will turn

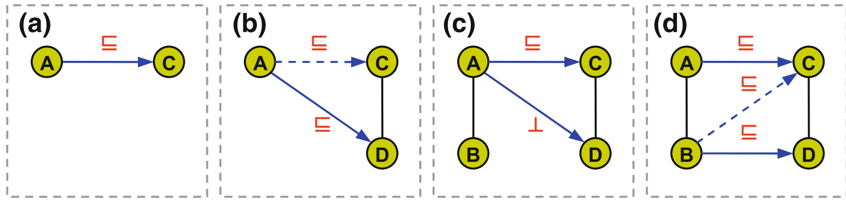


Fig. 8 Examples of applications of the **TreeSubsumedBy**

Fig. 9 Pseudo-code to compute a mapping element

```
function mapping ComputeMappingElement(node n1, node n2)
 {isLG, isMG: boolean;
  if ((<n1,n2,⊥> ∈ M) || IsRedundant(<n1,n2,⊥>)) then return <n1,n2,⊥>;
  if (<n1,n2,≡> ∈ M) then return <n1,n2,≡>;
  if ((<n1,n2,⊑> ∈ M) || IsRedundant(<n1,n2,⊑>)) then isLG := true;
  if ((<n1,n2,⊒> ∈ M) || IsRedundant(<n1,n2,⊒>)) then isMG := true;
  if (isLG && isMG) then return <n1,n2,≡>;
  if (isLG) then return <n1,n2,⊑>;
  if (isMG) then return <n1,n2,⊒>;
  return NULL;
 };
function boolean IsRedundant(mapping <n1,n2,R>)
 {switch (R)
   {case ⊑: if (VerifyCondition1(n1,n2)) then return true; break;
    case ⊒: if (VerifyCondition2(n1,n2)) then return true; break;
    case ⊥: if (VerifyCondition3(n1,n2)) then return true; break;
    case ≡: if (VerifyCondition1(n1,n2) &&
                VerifyCondition2(n1,n2)) then return true;
   };
  return false;
 };
function boolean VerifyCondition1(node n1, node n2)
 {c1,c2: node;
  foreach c1 in Path(n1) do
    foreach c2 in SubTree(n2) do
      if ((<c1,c2,⊑> ∈ M) || (<c1,c2,≡> ∈ M)) then return true;
  return false;
 };
```

out to hold. We achieve this purpose by processing both trees top down. To maximize the performance of the system, **TreeMatch** has therefore been built as the sequence of three function calls: the first call to **TreeDisjoint** (line 80) computes the minimal set of disjointness mapping elements, while the second and the third call to **TreeSubsumedBy** compute the minimal set of subsumption mapping elements in the two directions modulo equivalence (lines 90–120). Notice that in the second call, **TreeSubsumedBy** is called with the input ontologies with swapped roles. The variable **direction** is used to change the direction of the subsumption. These three calls correspond to Step 1 above. They enforce patterns (1), (2), and (3). Line 130 in the pseudo-code of **TreeMatch** implements Step 2 and it will be described in the next subsection. It enforces pattern (4), as the combination of patterns (1) and (2).

TreeDisjoint (Fig. 6) is a recursive function that finds all disjointness minimal elements between the two sub-trees rooted in n1 and n2. Following the definition of redundancy, it basically searches for the first disjointness element along any pair of paths in the two input trees. Exploiting the nested recursion of **NodeTreeDisjoint** inside **TreeDisjoint**, for any node n1 in T1 (traversed top down, depth first) **NodeTreeDisjoint** visits all of T2, again top down, depth first. **NodeTreeDisjoint** (called at line 30, starting at line 60) keeps fixed the source node n1 and iterates on the whole target sub-tree below n2 till, for each path, the highest disjointness element, if any, is found. Any such disjointness element is added only if minimal (lines 90–120). The condition at line 80 is necessary and sufficient for redundancy. The idea here is to exploit the fact that any two nodes below two nodes involved in a disjointness mapping element are part of a redundant element and, therefore, to stop the recursion. This saves a lot of time expensive calls ($n * m$ calls with n and m the number of the nodes in the two sub-trees). Notice that this check needs to be performed on the full path. At this purpose, **NodeDisjoint**

Table 1 Complexity of the datasets

#	Dataset pair	Node count	Max depth	Average branching factor
1	Cornell/Washington	34/39	3/3	5.50/4.75
2	Topia/Icon	542/999	2/9	8.19/3.66
3	Source/Target	2,857/6,628	11/15	2.04/1.94
4	Eclass/Unspsc	3,358/5,293	4/4	3.18/9.09

checks whether the formula obtained by the conjunction of the formulas associated to the nodes n1 and n2 is unsatisfiable (lines 150–170).

TreeSubsumedBy (Fig. 7) recursively finds all minimal mapping elements where the strongest relation between the nodes is $\sqsubseteq$ (or dually, $\sqsupseteq$ in the second call; in the following we will concentrate only on the first call). Notice that **TreeSubsumedBy** assumes that the minimal disjointness elements are already computed. As a consequence, at line 30 it checks whether the mapping element between the nodes n1 and n2 is already in the minimal set. If this is the case it stops the recursion. This allows computing the stronger disjointness relation rather than subsumption when both hold (namely with an inconsistent node). Given n2, lines 40–50 implement a depth first recursion in the first tree till a subsumption is found. The test for subsumption is performed by function **NodeSubsumedBy** that checks whether the formula obtained by the conjunction of the formulas associated to the node n1 and the negation of the formula for n2 is unsatisfiable (lines 170–190). Lines 60–140 implement what happens after the first subsumption is found. The key idea is that, after finding the first subsumption, **TreeSubsumedBy** keeps recursing down the second tree till it finds the last subsumption. When this happens, the resulting mapping element is added to the minimal mapping (line 100). Notice that both **NodeDisjoint** and **NodeSubsumedBy** call the function **Unsatisfiable** which embeds a call to a SAT solver.

To fully understand **TreeSubsumedBy,** the reader should check what happens in the four situations in Fig. 8. In case (a) the first iteration of the **TreeSubsumedBy** finds a subsumption between A and C. As C has no children, it skips lines 80–90 and directly adds the mapping element $\langle A, C, \sqsubseteq\rangle$ to the minimal set (line 100). In case (b), as there is a child D of C the algorithm iterates on the pair A–D (lines 80–90) finding a subsumption between them. As there are no other nodes under D, it adds the mapping element $\langle A, D, \sqsubseteq\rangle$ to the minimal set and returns true. Therefore, LastNodeFound is set to true (line 90) and the mapping element between the pair A–C is recognized as redundant. Case (c) is similar. The difference is that **TreeSubsumedBy** will return false when checking the pair A–D (line 30)—thanks to previous computation of minimal disjointness mapping elements—and therefore the mapping element $\langle A, C, \sqsubseteq\rangle$ is recognized as minimal. In case (d) the algorithm iterates after the second subsumption mapping element is identified. It first checks the pair A–C and iterates on A–D concluding that subsumption does not hold between them (line 40). Therefore, it recursively calls **TreeSubsumedBy** between B and D. In fact, as $\langle A, C, \sqsubseteq\rangle$ will be recognized as minimal, it is not worth checking $\langle B, C, \sqsubseteq\rangle$ because of pattern (1). As a consequence the mapping element $\langle B, D, \sqsubseteq\rangle$ is recognized as minimal together with $\langle A, C, \sqsubseteq\rangle$.

Five observations about the **TreeMatch** function. The first is that, even if, overall, **TreeMatch** implements three loops instead of one, the wasted (linear) time is largely counterbalanced by the exponential time saved by avoiding a lot of useless calls to the SAT solver. The second is that, when the input trees T1 and T2 are two nodes, **TreeMatch** behaves as a node matching function which returns the semantic relation holding between the input nodes. The third is that the call to **TreeDisjoint** before the two calls to **TreeSubsumedBy** allows us to implement the partial order on relations defined in the previous section. In particular it allows returning only a disjointness mapping element when both disjointness and subsumption hold. The fourth is the fact that skipping (in the body of the **TreeDisjoint**) the two sub-trees where disjointness holds is what allows not only implementing the partial order (see the previous observation) but also saving a lot of useless calls to the node matching functions. The fifth and last observation is that the implementation of **TreeMatch** crucially depends on the fact that the minimal elements of the two directions of subsumption and disjointness can be computed independently (modulo inconsistencies).

4.2 Step 2: Computing the minimal mapping

The output of Step 1 is the set of all disjointness and subsumption mapping elements which are minimal modulo equivalence. The final step toward computing the minimal mapping is that of collapsing any two subsumption relations, in the two directions, holding between the same two nodes into a single equivalence relation. The key point is that equivalence is in the minimal set only if both subsumptions are in the minimal set. We have three possible situations:

1. None of the two subsumptions is minimal (in the sense that it has not been computed as minimal in Step 1): nothing changes and neither subsumption nor equivalence is memorized as minimal;
2. Only one of the two subsumptions is minimal while the other is not minimal (again according to Step 1): this case is solved by keeping only the subsumption mapping as minimal. Of course, during Step 3 (see below) the necessary computations will have to be done in order to show to the user the existence of an equivalence relation between the two nodes;

Table 2 Mapping sizes

#	S-Match	MinSMatch		
	Total mapping elements (t)	Total mapping elements (t)	Minimal mapping elements (m)	Reduction (%)
1	223	223	36	83.86
2	5,491	5,491	243	95.57
3	282,638	282,648	30,956	89.05
4	39,590	39,818	12,754	67.97

Table 3 Run time and SAT problems

#	Run time (ms)			SAT calls		
	S-Match	Min S-Match	Reduction (%)	S-Match	Min S-Match	Reduction (%)
1	472	397	15.88	3978	2273	42.86
2	141,040	67,125	52.40	1,624,374	616,371	62.05
3	3,593,058	1,847,252	48.58	56,808,588	19,246,095	66.12
4	6,440,952	2,642,064	58.98	53,321,682	17,961,866	66.31

Table 4 NALT and LCSH branches matched

Id	Source	Branch	Number of nodes	Enriched nodes (%)
A	NALT	Chemistry and physics	3,944	97
B	NALT	Natural resources, earth and environmental sciences	1,546	96
C	LCSH	Chemical elements	1,161	97
D	LCSH	Chemicals	1,372	93
E	LCSH	Management	1,137	91
F	LCSH	Natural resources	1,775	74

3. Both subsumptions are minimal (according to Step 1): in this case the two subsumptions can be deleted and substituted with a single equivalence element. This implements the fact that pattern (4) is exactly the combination of the patterns (1) and (2).

Note that Step 2 can be computed very easily in time linear with the number of mapping elements output of Step 1: it is sufficient to check for all the subsumption elements of opposite direction between the same two nodes and to substitute them with an equivalence element. This is performed by function **TreeEquiv** in Fig. 5.

4.3 Step 3: Computing the mapping of maximum size

We concentrate on the following problem: given two lightweight ontologies T1 and T2 and the minimal mapping M compute the mapping element between two nodes n1 in T1 and n2 in T2 or the fact that no element can be computed given the current available background knowledge. Other problems are trivial variations of this one. The corresponding pseudocode is given in Fig. 9.

ComputeMappingElement is structurally very similar to the NodeMatch function described in [12], modulo the key difference that no calls to SAT are needed. **ComputeMappingElement** always returns the mapping element with strongest relation. More in detail, a mapping element is returned by the algorithm either in the case it is in the minimal set or it is redundant w.r.t. an element in the minimal set. The check is performed according to the partial order enforced on the semantic relations, i.e., at line 30 for $\perp$, 40 for $\equiv$, 50 for $\sqsubseteq$ and 60 for $\sqsupseteq$, where the first condition checks for the presence of the mapping element in the minimal set and the second for redundancy. Note that if both $\sqsubseteq$ and $\sqsupseteq$ hold, equivalence is returned (line 80). The test for redundancy performed by **IsRedundant** reflects the definition of redundancy provided in Sect. 3 above. We provide only the code which does the check for the first pattern; the others are analogous. Given for example a mapping element $\langle$n1, n2, $\sqsubseteq\rangle$, condition 1 is verified by checking (line 250) whether in M there is an element $\langle$c1, c2, $\sqsubseteq\rangle$ or $\langle$c1, c2, $\equiv\rangle$

Table 5 Results of matching experiments

Matching experiment	Mapping of maximum size	Minimal mapping	Reduction (%)
A vs. C			
Mapping elements found	17,716	7,541	57.43
Disjointness	8,367	692	91.73
Equivalence	0	0	–
More general	0	0	–
More specific	9,349	6,849	26.74
A vs. D			
Mapping elements found	139,121	994	99.29
Disjointness	121,511	754	99.38
Equivalence	0	0	–
More general	0	0	–
More specific	17,610	240	98.64
A vs. E			
Mapping elements found	9,579	1,254	86.91
Disjointness	9,579	1,254	86.91
Equivalence	0	0	–
More general	0	0	–
More specific	0	0	–
B vs. F			
Mapping elements found	27,191	1,232	95.47
Disjointness	21,352	1,141	94.66
Equivalence	24	1	95.83
More general	2,808	30	98.93
More specific	3,007	60	98.00

Table 6 Precision and recall for minimized, normal, and maximized sets

Dataset pair	Precision (%)			Recall (%)		
	Min	Norm	max	Min	Norm	Max
101/304	32.47	9.75	69.67	86.21	93.10	92.79
Topia/icon	16.87	4.86	45.42	10.73	20.00	42.11
Source/target	74.88	52.03	48.40	10.35	40.74	53.30

with c1 ancestor of n1 (c1 is taken from the path of n1, line 230) and c2 descendant of n2 (c2 is taken from the sub-tree of n2, line 240). Notice that **ComputeMappingElement** calls **IsRedundant** at most three times and, therefore, its computation time is linear with the number of mapping elements in M.

5 Evaluation

The algorithm presented in the previous section has been implemented by taking the node matching routines of the state of the art matcher S-Match [12] and by changing the way the tree structure is matched. We call the new matcher **MinSMatch**. The evaluation has been performed by directly comparing the results of MinSMatch and S-Match on several real-world datasets. All tests have been performed on a Pentium D 3.40 GHz with 2 GB of RAM running Windows XP SP3 operating system with no additional applications running except the matching system. Both systems were limited to allocating no more than 1GB of RAM. The tuning parameters were set to the default values. The selected datasets had been already used in previous evaluations, see [3,13]. Some of these datasets can be found at the Ontology Alignment Evaluation Initiative (OAEI) web site[2]. The first two datasets describe courses and will be called **C**ornell and **W**ashington, respectively. The second two come from the arts domain and will be referred to as **T**opia and **I**con, respectively. The third two datasets have been extracted from the Looksmart, Google and Yahoo! directories and will be referred to as **S**ource and **T**arget. The fourth two datasets contain portions of the two business directories eCl@ss[3] and UNSPSC[4] and will be referred to as **E**class and **U**nspsc. Table 1 describes some indicators of the complexity of these datasets.

Consider Table 2. The reduction in the last column is calculated as $(1 - m/t)$, where m is the number of elements in the minimal set and t is the total number of elements in the mapping of maximum size, as computed by MinSMatch. As it can be easily noticed, we have a significant reduction, in the range 68–96%.

The second interesting observation is that in Table 2, in the last two experiments, the number of total mapping elements computed by MinSMatch is slightly higher (compare the second and the third column). This is due to the fact that in the presence of one of the patterns, MinSMatch directly infers the existence of a mapping element without testing it. This allows MinSMacth, differently from S-Match, to reduce missing elements because of failures of the node matching functions (because of lack of background knowledge [11]). One such example from our experiments is reported below (directories Source and Target):

\ Top\Computers\Internet\Broadcasting\Video Shows
\ Top\Computing\Internet\Fun & Games\Audio & Video\Movies

We have a minimal mapping element which states that Video Shows $\sqsupseteq$ Movies. The element generated by this minimal one, which is captured by MinSMatch and missed by S-Match (because of the lack of background knowledge about the relation between "Broadcasting" and "Movies") states that Broadcasting $\sqsupseteq$ Movies.

To conclude our analysis, Table 3 shows the reduction in computation time and calls to SAT. As it can be noticed, the time reductions are substantial, in the range 16–59%, but where the smallest savings are for very small ontologies. In principle, the deeper the ontologies the more we should save. The interested reader can refer to [12,3] for a detailed qualitative and performance evaluation of S-Match w.r.t. other state of the art matching algorithms.

These results are confirmed by a recent experiment we conducted in matching NALT with LCSH [16] that are much deeper in structure (15 and 25 levels, respectively). We executed MinSMatch on a selection of the branches which turned out to have a high percentage of semantically enrichable nodes, i.e., nodes whose labels could be parsed with success by our NLP pipeline. See Table 3 for details. Table 4 shows evaluation details about conducted experiments in terms of the branches which are matched, the number of elements in the mapping of maximum size (obtained by propagation from the elements in the minimal mapping), the number of elements in the minimal mapping and the percentage of reduction in the size of the minimal set w.r.t. the size of the mapping of maximum size.

We ran MinSMatch both between branches with an evident overlap in the topic (i.e., A vs. C and D, B vs. F) and between clearly unrelated branches (i.e., A vs. E). As expected, in the latter case we obtained only disjointness relations. This demonstrates that the tool is able to provide clear hints of places in which it is not worth to look at in case of search and navigation. All experiments confirm that the minimal mapping contains significantly less elements w.r.t. the mapping of maximum size (from 57 to 99%). Experiments also show that exact equivalence is quite rare. We found just 24 equivalences, and only one in a minimal mapping. This phenomenon has been observed also in other projects, for instance in Renardus [18] and CARMEN[5].

Yet it is clear that, even if faster, automatic approaches require manual validation and augmentation of computed mappings (see for instance [9], which also describes a tool that supports this task). This is exactly where minimal mappings can be of great help. In fact, regardless the quality of the mapping returned by the algorithm, during the validation phase we can concentrate on the elements in the minimal mapping, a very small portion of the complete mapping, i.e., the mapping of maximum size. This makes manual validation much easier, faster, and less error-prone. In [22], we give concrete suggestions on how to effectively conduct the validation process.

[2] http://oaei.ontologymatching.org/2006/directory/.

[3] http://www.eclass-online.com/.

[4] http://www.unspsc.org/.

[5] http://www.bibliothek.uni-regensburg.de/projects/carmen12/index.html.

We also conducted several experiments (described extensively in [2]) to study the differences between precision and recall measures when comparing the minimized (the mapping obtained by removing redundant correspondences) and maximized (the mapping obtained by adding all the redundant correspondences) versions of the golden standards with the minimized and maximized versions of the mapping returned by S-Match [12]. We used three different golden standards [1] already used in several evaluations. The first two datasets (101 and 304) come from OAEI; the two ontologies describe publications, contain few nodes and corresponding golden standard is exhaustive. It only contains equivalence correspondences. The second (Topia and Icon) and third (Source and Target) pairs are described in Table 1.

Table 6 shows precision and recall figures obtained from the comparison of the minimized mapping with the minimized golden standards (min), the original mapping with the original golden standards (norm) and the maximized mapping with the maximized golden standards (max), respectively. For what said above, the max columns provide the most accurate results. As it can be noted from the measures obtained comparing the maximized versions with the original versions, the performance of the S-Match algorithm is on average better than expected (with the exception of the precision figure of the Source/Target experiment). More in general, the quality of the mapping is a function of the quantity and quality of the available background knowledge. We are currently developing new methodologies for its construction [8].

6 Conclusions

In this article, we have provided a definition and a fast algorithm for the computation of the minimal mapping between

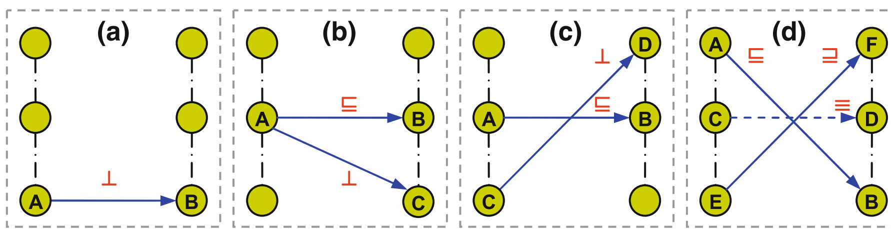


Fig. 10 Some trivial cases which do not fall in the redundancy patterns

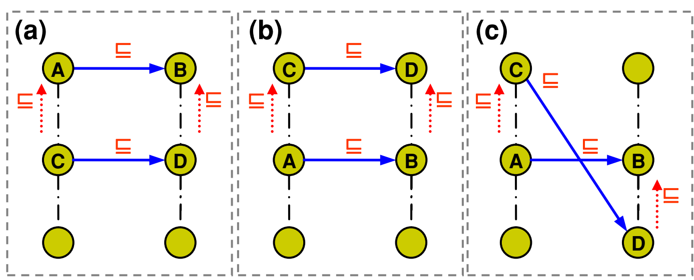


Fig. 11 Completeness of condition (1)

	$A \notin path(C)$	$D \notin path(B)$	Rationale
(a)	NO	YES	$C \sqsubseteq \ldots \sqsubseteq A$, $D \sqsubseteq \ldots \sqsubseteq B$, $A \sqsubseteq B$ cannot derive $C \sqsubseteq D$
(b)	YES	NO	$A \sqsubseteq \ldots \sqsubseteq C$, $B \sqsubseteq \ldots \sqsubseteq D$, $A \sqsubseteq B$ cannot derive $C \sqsubseteq D$
(c)	YES	YES	$A \sqsubseteq \ldots \sqsubseteq C$, $D \sqsubseteq \ldots \sqsubseteq B$, $A \sqsubseteq B$ cannot derive $C \sqsubseteq D$

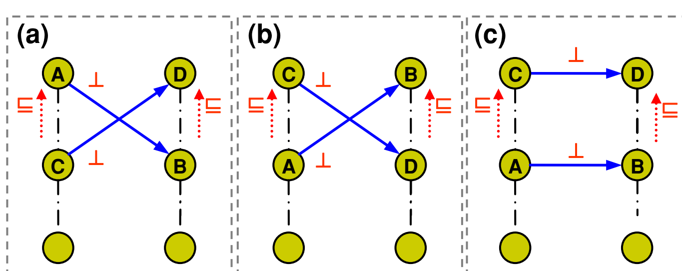


Fig. 12 Completeness of condition (3)

	$A \notin path(C)$	$B \notin path(D)$	Rationale
(a)	NO	YES	$C \sqsubseteq \ldots \sqsubseteq A$, $B \sqsubseteq \ldots \sqsubseteq D$, $A \perp B$ cannot derive $C \perp D$
(b)	YES	NO	$A \sqsubseteq \ldots \sqsubseteq C$, $D \sqsubseteq \ldots \sqsubseteq B$, $A \perp B$ cannot derive $C \perp D$
(c)	YES	YES	$A \sqsubseteq \ldots \sqsubseteq C$, $D \sqsubseteq \ldots \sqsubseteq B$, $A \perp B$ cannot derive $C \perp D$

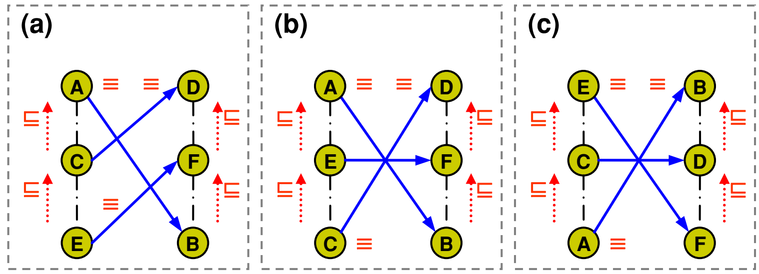


Fig. 13 Completeness of condition (4)

	A ∉ path(C)	D ∉ path(B)	C ∉ path(E)	F ∉ path(D)	Rationale
(a)	NO	NO	NO	YES	E ⊑ … ⊑ C, C ⊑ … ⊑ A, B ⊑ … ⊑ F, F ⊑ … ⊑ D, A ≡ B and E ≡ F cannot derive C ≡ D (we can only derive C ⊑ D).
(b)	NO	NO	YES	YES	C ⊑ … ⊑ E, E ⊑ … ⊑ A, B ⊑ … ⊑ F, F ⊑ … ⊑ D, A ≡ B and E ≡ F cannot derive C ≡ D (we can only derive C ⊑ D).
…					
(c)	YES	YES	YES	YES	Covered by condition (4) inverting the roles of m and m''

two lightweight ontologies. The evaluation shows a substantial improvement in the (much lower) computation time, in the (much lower) number of elements which need to be stored and handled and in the (higher) total number of mapping elements which are computed. As part of the evaluation, we have presented the results of a matching experiment (Table 5) we conducted between two large-scale knowledge organization systems: NALT and LCSH. They confirm that the minimal mapping always contains a very little portion of the overall number of correspondences between the two ontologies; this makes manual validation much easier, up to two orders of magnitude faster, and less error-prone.

Finally, we have shown that to obtain more accurate evaluations one should maximize both the golden standard and the matching result. Experiments show that for instance the state of the art matcher S-Match performs on average better than expected.

As future study, we plan to explore the possibility to extend the notion of minimal mapping from lightweight ontologies to generic ontologies.

Appendix: proofs of the theorems

Theorem 1 (Redundancy, soundness and completeness) *Given a mapping M between two lightweight ontologies O_1 and O_2, a mapping element $m' \in M$ is logically redundant w.r.t. another mapping element $m \in M$ if and only if it satisfies one of the conditions of Definition 3.*

Proof *Soundness:* The argumentation provided in Sect. 3 as a rationale for the patterns already provides a full demonstration for soundness.

Completeness: We can demonstrate the completeness by showing that we cannot have redundancy (w.r.t. another mapping element) in the cases which do not fall in the conditions listed in Definition 3. We proceed by enumeration, negating each of the conditions. There are some trivial cases we can exclude in advance:

- The trivial case in which m′ is the only mapping element between the lightweight ontologies. See Fig. 10a;
- Incomparable symbols. The only cases of dependency across symbols are captured by conditions (1) and (2) in Definition 3, where equivalence can be used to derive the redundancy of a more or less specific mapping element. This is due to the fact that equivalence is exactly the combination of more and less specific. No other symbols can be expressed in terms of the others. This means for instance that we cannot establish implications between an element with more specific and one with disjointness. In Fig. 10b, the two elements do not influence each other;
- All the cases of inconsistent nodes. See for instance Fig. 10 c. If we assume the element ⟨A, B, ⊑⟩ to be correct, then according to pattern (1) the mapping element between C and D should be ⟨C, D, ⊑⟩. However, in case of inconsistent nodes the stronger semantic relation ⊥ holds. The algorithm presented in Sect. 4 correctly returns ⊥ in these cases;

- Cases of underestimated strength not covered by the previous cases, namely the cases in which equivalence holds instead of the (weaker) subsumption. Look for instance at Fig. 10d. The two subsumptions in $\langle A, B, \sqsubseteq\rangle$ and $\langle E, F, \sqsubseteq\rangle$ must be equivalences. As a consequence, $\langle C, D, \equiv\rangle$ is redundant for pattern (4). In fact, the chain of subsumptions $E \sqsubseteq \ldots \sqsubseteq C \sqsubseteq \ldots \sqsubseteq A \sqsubseteq B \sqsubseteq \ldots \sqsubseteq D \sqsubseteq \ldots \sqsubseteq F$ allows to conclude that $E \sqsubseteq F$ holds and therefore $E \equiv F$. Symmetrically, we can conclude that $A \equiv B$. Note that the mapping elements $\langle A, B, \sqsubseteq\rangle$ and $\langle E, F, \sqsupseteq\rangle$ are minimal. We identify the strongest relations by propagation (at step 3 of the proposed algorithm, as described in Sect. 4).

We refer to all the other cases as the *meaningful cases*.

Condition (1): its negation is when $R \neq$ "$\sqsubseteq$" or $A \notin$ path(C) or $D \notin$ path(B). The cases in which R = "$\sqsubseteq$" are shown in Fig. 11. For each case, the provided rationale shows that available axioms cannot be used to derive $C \sqsubseteq D$ from $A \sqsubseteq B$. The remaining meaningful cases, namely only when R = "$\equiv$", are similar.

Condition (2): it is the dual of condition (1).

Condition (3): its negation is when $R \neq$ "$\perp$" or $A \notin$ path(C) or $B \notin$ path(D). The cases in which R = "$\perp$" are shown in Fig. 12. For each case, the provided rationale shows that available axioms cannot be used to derive $C \perp D$ from $A \perp B$. There are no meaningful cases for $R \neq$ "$\perp$".

Condition (4): it can be easily noted from Fig. 3 that the redundant elements identified by pattern (4) are exactly all the mapping elements $m' = \langle C, D, \equiv\rangle$ with source C and target D, respectively, between (or the same of) the source node and target node of two different mapping elements $m = \langle A, B, \equiv\rangle$ and $m'' = \langle E, F, \equiv\rangle$. This configuration allows to derive from m and m'' the subsumptions in the two directions which amount to the equivalence. The negation of condition 4 is when $R \neq$ "$\equiv$" in m or m'' or $A \notin$ path(C) or $D \notin$ path(B) or $C \notin$ path(E) or $F \notin$ path(D). In almost all the cases (14 over 15) in which R = "$\equiv$" we just move the source C or the target D outside these ranges. For sake of space we show only some of such cases in Fig. 13. The rationale provided for cases (a) and (b) shows that we cannot derive $C \equiv D$ from $A \equiv B$ and $E \equiv F$. The only exception (the remaining 1 case over 15), represented by case (c), is when $A \notin$ path(C) and $D \notin$ path(B) and $C \notin$ path(E) and $F \notin$ path(D). This case, however, is covered by condition 4 by inverting the role of m and m''. The remaining cases, namely when $R \neq$ "$\equiv$" in m or m'', are not meaningful.

This completes the demonstration. □

References

1. Autayeu, A., Giunchiglia, F., Andrews, P., Ju, Q.: Lightweight parsing of natural language metadata. In: Proceedings of the First Natural Language Processing for Digital Libraries Workshop (2009)
2. Autayeu, A., Maltese, V., Andrews, P.: Recommendations for better quality ontology matching evaluations. In: Proceedings of the 2nd AISB Workshop on Matching and Meaning (2010)
3. Avesani, P., Giunchiglia, F., Yatskevich, M.: A large scale taxonomy mapping evaluation. In: Proceedings of International Semantic Web Conference (ISWC 2005), pp. 67–81 (2005)
4. Borgida, A., Serafini, L.: Distributed description logics: assimilating information from peer sources. J. Data Semant. **1**, 153–184
5. Caracciolo, C., Euzenat, J., Hollink, L., Ichise, R., Isaac, A., Malaisé, V., Meilicke, C., Pane, J., Shvaiko, P.: Results of the ontology alignment evaluation initiative 2008. In: The 7th International Semantic Web Conference, vol. 431(2), pp. 73–118 (2008)
6. David, J., Euzenat, J.: On fixing semantic alignment evaluation measures. In: Proceedings of the Third International Workshop on Ontology Matching, pp. 25–36 (2008)
7. Doerr, M.: Semantic problems of thesaurus mapping. J. Digit. Inf. **1**(8), article 52 (2001)
8. Dutta, B., Giunchiglia, F., Maltese, V.: A facet-based methodology for geo-spatial modelling. In: Proceedings of GEOS, 6631, pp. 133–150 (2011)
9. Falconer, S., Storey, M.: A cognitive support framework for ontology mapping. In: Proceedings of ISWC/ASWC (2007)
10. Giunchiglia, F., Marchese, M., Zaihrayeu, I.: Encoding classifications into lightweight ontologies. J. Data Semant. **8**, 57–81 (2006)
11. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: Discovering missing background knowledge in ontology matching. In: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), pp. 382–386 (2006)
12. Giunchiglia, F., Yatskevich, M., Shvaiko, P.: Semantic matching: algorithms and implementation. J. Data Semant. **IX**, 1–38 (2007)
13. Giunchiglia, F., Yatskevich, M., Avesani, P., Shvaiko, P.: A large dataset for the evaluation of ontology matching systems. Knowl. Eng. Rev. J. **24**(2), 137–157 (2008)
14. Giunchiglia, F., Zaihrayeu, I.: Lightweight ontologies. In: The Encyclopedia of Database Systems. Springer, Berlin (2008, to appear)
15. Giunchiglia, F., Maltese, V., Autayeu, A.: Computing minimal mappings. At the 4th Ontology Matching Workshop as part of the ISWC (2009)
16. Giunchiglia, F., Soergel, D., Maltese, V., Bertacco, A.: Mapping large-scale Knowledge Organization Systems. In: Proceedings of the 2nd International Conference on the Semantic Web and Digital Libraries (ICSD) (2009)
17. Hjørland, B.: What is knowledge organization (KO)? Knowledge organization. Int. J. Devot. Concept Theory Classif Index. Knowl. Represent. **35**(2/3), 86–101 (2008)
18. Koch, T., Neuroth, H., Day, M.: Renardus: cross-browsing European subject gateways via a common classification system (DDC). In: McIlwaine, I.C. (ed.) Subject retrieval in a networked environment. Proceedings of the IFLA satellite meeting held in Dublin, pp. 25–33 (2003)
19. Kovács, L., Micsik, A.: Extending semantic matching towards digital library contexts. In: Proceedings of the 11th European Conference on Digital Libraries (ECDL), pp. 285–296 (2007)
20. Lauser, B., Johannsen, G., Caracciolo, C., Keizer, J., van Hage, W.R., Mayr, P.: Comparing human and automatic thesaurus mapping approaches in the agricultural domain. In: Proceedings of the International Conference on Dublin Core and Metadata Applications, pp. 43–53 (2008)

21. Madhavan, J., Bernstein, P.A., Domingos, P., Halevy, A.Y.: Representing and reasoning about mappings between domain models. At the 18th National Conference on Artificial Intelligence (AAAI 2002), pp. 80–86 (2002)
22. Maltese, V., Giunchiglia, F., Autayeu, A.: Save up to 99% of your time in mapping validation. In: Proceedings of the 9th International ODBASE conference (2010)
23. Marshall, B., Madhusudan, T.: Element matching in concept maps. In: Proceeidngs of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2004), pp. 186–187 (2004)
24. Meilicke, C., Stuckenschmidt, H., Tamilin, A.: Improving automatically created mappings using logical reasoning. In: Proceedings of the 1st International Workshop on Ontology Matching OM-2006, CEUR Workshop Proceedings, vol. 225 (2006)
25. Meilicke, C., Stuckenschmidt, H., Tamilin, A.: Reasoning support for mapping revision. J. Logic Comput. (2008). doi:10.1093/logcom/exn047
26. Nicholson, D., Dawson, A., Shiri, A.: HILT: a pilot terminology mapping service with a DDC spine. Cat. Classif. Quart. **42**(3/4), 187–200 (2006)
27. O'Neill, E., Chan, L.: FAST (Faceted Application for Subject Technology): a simplified LCSH-based vocabulary. World Library and Information Congress: 69th IFLA General Conference and Council, 1–9 August, Berlin (2003)
28. Shvaiko, P., Euzenat, J.: Ontology Matching. Springer, Secaucus, NJ (2007)
29. Shvaiko, P., Euzenat, J.: Ten challenges for ontology matching. In: Proceedings of the 7th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE) (2008)
30. Soergel, D.: A universal source thesaurus as a classification generator. J. Am. Soc. Inf. Sci. **23**(5), 299–305 (1972)
31. Stuckenschmidt, H., Serafini, L., Wache, H.: Reasoning about ontology mappings. In: Proceedings of the ECAI-06 Workshop on Contextual Representation and Reasoning (2006)
32. Vizine-Goetz, D., Hickey, C., Houghton, A., Thompson, R.: Vocabulary mapping for terminology services. J. Digit. Inf. **4**(4) (2004)
33. Whitehead, C.: Mapping LCSH into Thesauri: the AAT Model. In: Beyond the Book: Extending MARC for Subject Access, p. 81 (1990)
34. Zaihrayeu, I., Sun, L., Giunchiglia, F., Pan, W., Ju, Q., Chi, M., Huang, X.: From web directories to ontologies: natural language processing challenges. In: 6th International Semantic Web Conference (ISWC 2007) (2007)
35. Zeng, M.L., Chan, L.M.: Trends and issues in establishing interoperability among knowledge organization systems. J. Am. Soc. Inf. Sci. Technol. **55**(5), 377–395 (2004)