Optimised Classification for Taxonomic Knowledge Bases

Dmitry Tsarkov and Ian Horrocks
University of Manchester, Manchester, UK
{tsarkov|horrocks}@cs.man.ac.uk

Abstract

Many legacy ontologies are now being translated into Description Logic (DL) based ontology languages in order to take advantage of DL based tools and reasoning services. The resulting DL Knowledge Bases (KBs) are typically of large size, but have a very simple structure, i.e., they consist mainly of shallow taxonomies. The classification algorithms used in state-of-the-art DL reasoners may not deal well with such KBs In this paper we propose an optimisation which dramatically speeds-up classification for such KBs.

1 Introduction

Motivated by the W3C recommendation, and the availability of DL based tools and reasoning services, many legacy ontologies are now being translated into the Description Logic (DL) based OWL DL ontology language [3]. The resulting DL Knowledge Bases (KBs) are typically of (very) large size, but often have only a (very) simple structure. In particular, the sub-class hierarchy often resembles a taxonomy (i.e., an asserted hierarchy of primitive concepts), and is very broad and shallow.

The classification algorithms used in state-of-the-art DL reasoners (such as FaCT and RACER) may not deal well with such KBs, mainly due to the fact that some classes in the hierarchy will have very large numbers of direct sub-classes. For such KBs, the top-down phase of the standard classification algorithm [2] will perform a very large number of subsumption tests, almost all of which will fail (i.e., conclude that no subsumption relationship holds). Although the simple structure of the KB means that the time required for each such test is small, the very large number of tests can still lead to performance problems.

In this paper we present a classification optimisation that identifies (subsets of) KBs for which it is possible to compute the concept hierarchy without performing *any* subsumption tests. This technique is very effective when used on the kinds of legacy KB described above as it avoids performing large numbers

of negative subsumption tests. It also turns out to produce (smaller) improvements in classification times for some more complex ontologies, as even in these ontologies a significant part of the upper level structure often resembles a taxonomy.

The drawback of this new optimisation is that it is only applicable to purely "definitional" knowledge bases, i.e., those that do not contain any general concept inclusion axioms (GCIs). However, the well known *absorption* optimisation has already been shown to be able to eliminate GCIs from typical ontologies [6] and, moreover, when absorption fails to eliminate GCIs, we may already expect serious reasoning performance problems from other sources (i.e., the hardness of individual subsumption tests). It is also only applicable to that portion of a KB which has the necessary simple structure, but this is often a major part of legacy KBs and, as mentioned above, is usually a significant part of more complex KBs.

2 Preliminaries

Description Logics are concept (class) based knowledge representation systems. They are (usually) decidable fragments of First Order Predicate Calculus, and have a have standard first order style model theoretic semantics [1]. The formal specification of semantics coupled with decidability allows for the design of decision procedures (sound, complete and terminating algorithms) for key reasoning tasks such as concept subsumption.

A DL Knowledge Base is often thought of as consisting of two parts: a *Tbox* and an *Abox*. The Tbox consists of a set of axioms that describe constraints on instances of given concepts (roughly akin to a conceptual schema in a database setting); the Abox consists of a set of axioms that assert instance relationships between individuals and concepts, and role relationships between pairs of individuals (roughly akin to data in a database setting). Tbox axioms are of the form $C \sqsubseteq D$ or $C \equiv D$, where C and D are concepts. When C is a concept name, such an axiom is often called a *definition* (of C), and when a definition is of the form $C \sqsubseteq D$, C is called a *primitive* concept.

Classification of a Tbox is the task of computing and caching the concept hierarchy for all of the named concepts that occur in the Tbox, i.e., computing the subsumption partial ordering of the named concepts.[1] Tbox classification is a basic reasoning task for DL reasoners—the concept hierarchy may be interesting in its own right, and is used to optimise many other reasoning tasks (e.g., query answering). This latter point is particularly important as, even if the Tbox part of a KB is very simple, the Abox may describe instances of complex concept expressions. This is typical, e.g., for applications of the Gene Ontology [4].

We will first briefly recall the optimised procedure for computing the concept

[1] We assume w.l.o.g. that all concept names occurring in the Abox also occur in the Tbox; this can be achieved by adding axioms of the form $C \sqsubseteq \top$ to the Tbox for any concept name C that would otherwise occur only in the Abox.

hierarchy first described in [2]. In this procedure, all concept names are sorted into *definitional order*, i.e. if concept name D occurs in the definition of concept name C, then $D \leq C$.[2] The concept hierarchy is initialised to contain the two concepts $\top$ and $\bot$ (with $\top$ being a super concept of $\bot$), and the named concepts are classified (added to the hierarchy in the appropriate position) one at a time in definitional order.

Classifying a concept involves two phases: a top-down phase in which its *parents* (i.e., direct subsumers) are computed, and a bottom-up phase in which its *children* (i.e., direct subsumees) are computed. In many cases, adding concepts in definitional order may allow the bottom-up phase to be omitted (because when a new concept is classified its only child will be $\bot$).

Various optimisations are used in order to minimise the number of subsumption tests needed in each phase. For example, when adding a concept C to the hierarchy, a top-down breadth first traversal is used that only checks if D subsumes C when it has already been determined that C is subsumed by all the concepts in the hierarchy that subsume D. The structure of Tbox axioms is also used to compute a set of *told subsumers* of C (i.e., trivially obvious subsumers). For example, if the Tbox contains an axiom $C \sqsubseteq D_1 \sqcap D_2$, then both D_1 and D_2, as well as all *their* told subsumers, are told subsumers of C. As subsumption is immediate for told subsumers, no tests need to be performed w.r.t. these concepts. In order to maximise the benefit of this optimisation, all of the told subsumers of a concept C are classified *before* C itself is classified.

The told subsumer optimisation can be used to approximate the position of C in the hierarchy: all of its told subsumers, and any super-concepts of these told-subsumers, can be marked as subsumers of C. The most specific concepts in this set of marked concepts are then candidates to be parents of C. In the standard algorithm, however, it is necessary to check (recursively) if the children of these concepts are also subsumers of C. When it has been determined for some subsumer D of C that none of the children of D subsume C, then we know that D is a parent of C.

At the end of the top-down phase we will have computed the set of parents of C; all of the concepts in this set, along with all their super-concepts, are subsumers of C; all other concepts are non-subsumers of C. The next step is to determine the set of children of C (as mentioned above, this step can be omitted for a primitive concept when concepts have been classified in definitional order [2]). This phase is very similar to (the reverse of) the top-down one, and as our optimisation only relates to the top-down phase we won't describe it here—interested readers can refer to [1] for full details.

For large and shallow taxonomies, a concept D may have hundreds or even thousands of children. If, when classifying a concept C, one of its told subsumers is D, then the above algorithm may lead to *all* of the other children of D being

[2] In the FaCT++ implementation we actually use *quasi-definitional order*, as proposed in [5], but to simplify the presentation we will assume that definitional order is used.

checked to see if they subsume C. Although the time taken for each such test may be small, the cumulative cost of all these tests may be prohibitive when classifying such a Tbox. Moreover, in many cases all of these tests will be negative (i.e., no subsumption relationship will be found), and might be thought of as somehow "wasted". The objective of our optimisation is to avoid these "wasted" tests.

3 Completely Defined Concepts

Given a Tbox $\mathcal{T}$, a primitive concept C is said to be completely defined w.r.t. $\mathcal{T}$ when, for the definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ in $\mathcal{T}$, it holds that:

1. For all $1 \leqslant i \leqslant n$, C_i is a primitive concept.
2. (minimality) There exist no $i \neq j$ such that $1 \leq i, j \leq n$ and $C_i \sqsubseteq C_j \sqcap \ldots$.

When the Tbox is obvious from the context we will talk about completely defined concepts without reference to the Tbox.

If we assume a cycle-free Tbox containing only CD concepts and no GCIs, then the classification process is very simple. In fact, we don't need to perform any subsumption tests at all: the position of every concept in the hierarchy is *completely defined* by its told subsumers. If concepts are processed in definitional order, then when a concept C is classified, where C is defined by the axiom $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, the parents of C are $C_1, \ldots, C_n$, and the only child of C is $\bot$. Note that every concept in such a taxonomy is satisfiable, because there is no use of negation.

The following theorem is straightforward:

Theorem 1 *If a cycle-free KB contains only completely-defined concepts and no general axioms, then the taxonomy built by the above method will be correct.*

This theorem is, however, of very little practical value due to the very stringent conditions on the structure of the Tbox. In the following we will show how the basic technique can be made more useful by weakening some of these conditions.

Primitivity. In general, a CD concept should not have non-primitive concepts in its definition. This is because, when the hierarchy already includes non-primitive concepts (which will be the case given definitional order classification), the bottom-up phase can not be omitted, and the CD method could therefore lead to incorrect results. Assume, e.g., a TBox

$$\{C \sqsubseteq C_1 \sqcap C_2 \sqcap C_3, \quad C' = C_1 \sqcap C_2\}. \tag{1}$$

Using the CD classification approach, C will be classified under C_1, C_2, C_3, whereas it should be classified under C' and C_3.

One case in which this condition can be weakened is for synonyms. A non-primitive concept C is a *synonym* if it's definition is of the form $C = D$, where

D is a primitive concept. Synonyms may came from an application domain, or occur as a result of KB simplification, KB merging, etc.

It is easy to see that synonyms don't require special classification: once D is classified, C will take the same place in the hierarchy. So, adding synonyms to the CD-only KB still allows application of the CD approach.

Minimality. Non-minimal concepts may occur as a result of badly designed ontologies and/or due to absorption of GCIs. The minimality check may, however, be removed from the definition of CD concepts in the classification algorithm. Indeed, checking if each C_i in a definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ is really a parent of C (i.e., has no children that are subsumers of C) is exactly the check that is needed in order to detect non-minimality. This check is relatively cheap and already exists in the classification algorithm.

Non-CD concepts. This is the most important case, because "interesting" ontologies, including most ontologies designed using DL based languages, will contain concept constructors other than conjunction, and this will lead to some concepts being non-CD. This means that, in its current form, the CD approach will usually be largely useless. On the other hand, almost all KBs will contain *some* CD concepts. In this case, it may be possible to split the Tbox into two parts—a CD part (i.e., containing only CD concepts) and a non-CD part—and use the CD algorithm only for the CD part.

Note that such a split will not introduce any problems if the CD part of the classification is performed first—in fact the classification of the CD part is independent of the non-CD part of the Tbox because the definitions of CD concepts only refer to other CD concepts. In the Tbox 1 above, for example, concepts C_1, C_2, C_3 and C will be in the CD part, and C' in the non-CD part. After CD-classification C will have 3 parents, and standard algorithm then insert C' with parents C_1, C_2 and child C.

Cycles. We will distinguish two kinds of definitional cycles. The first (and simplest) is a cycle via concept names, as in the Tbox $K = \{A \sqsubseteq B \sqcap C, B \sqsubseteq A\}$. This kind of cycle can be detected syntactically and transformed into an equivalent definition $K' = \{A \sqsubseteq C, B = A\}$ where A is a CD concept and B is a synonym of A.

Any other kind of terminological cycle must involve non-CD concepts, and so must occur in the non-CD part of TBox. In this case it will be dealt with in the normal way by the standard classification algorithm.

General axioms. GCIs are axioms of the form $C \sqsubseteq D$, where C and D are arbitrary concept expressions.[3] It is easy to see that, in the general case, the CD approach cannot be used in the presence of GCIs. Consider, for example, a Tbox $K = \{C \sqsubseteq \top, \top \sqsubseteq D\}$. In this case, the CD algorithm classifies C under $\top$, whereas it should be classified under D.

[3]Note that, in case there are multiple axioms of the form $C \sqsubseteq D$ or $C \equiv D$ for some concept name C, then only one of these can be considered the definition of C, and the rest must be treated as GCIs (or, in the case of $C \equiv D$, as a pair of GCIs $C \sqsubseteq D$ and $D \sqsubseteq C$).

Fortunately, most realistic KBs contain only general axioms that can be absorbed into either concept implications [6] or role domain restrictions [8], and in this case the CD approach is still applicable.

4 Two-stage Approach Using CD.

The two-stage CD classification algorithm has been implemented in our FaCT++ reasoner as follows. First of all, the following transformations are performed on the Tbox (only transformations relevant to the classification algorithm are mentioned here):

1. Absorb general axioms into concept definitions and/or role domains. If some of the axioms are not absorbable, set `useCD` to `false`. If all the axioms were absorbed, set `useCD` to `true`.

2. Transform simple cycles into sets of synonyms.

3. If `useCD` is `true`, mark some concepts as CD. Namely, $\top$ is marked as CD; a primitive concept C is marked as CD iff it has the definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$ and every C_i is marked CD; a non-primitive concept D is marked CD iff it has definition $D = C$ and C is marked CD.

If `useCD` is `true` after the preprocessing, the CD classifier is run prior to the general classifier. The CD classifier works on concepts that are marked CD, processing them in definitional order. For each such concept C, the steps it performs are as follows:

1. If C is a synonym of some already classified concept D, then insert C at the same place as D.

2. For CD C with definition $C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, concepts $C_1, \ldots, C_n$ are candidates to be parents of C.

3. For every candidate C_i, check whether it is redundant, i.e. whether C_i has a child that is an ancestor of a C. This can be done by labelling all ancestors of candidate concepts: labelled candidates will be redundant. Remove redundant candidates from the list of candidates.

4. Insert C into the taxonomy with the remaining candidates as parents and $\bot$ as the only child.

Then the rest of the ontology is then classified using the standard classification algorithm.

We have tested our implementation using several KBs: NCI is the National Cancer Research Institute ontology; GO is the Gene Ontology from the Gene Ontology Consortium; GALEN is the anatomical part of the well-known medical terminology ontology [7]. Details of KB characteristics are given in Table 1,

KB	PConcepts	nCD	NConcepts	Synonyms
NCI	27652	15195	0	0
GO	13926	11718	3	0
GALEN	2048	546	699	18

Table 1: test KB properties

KB	CD	time	nOps	nTests	nCache
NCI	no	76.40	1,614,903	0	10,311,489
	yes	3.61	1,012,281	0	766,054
GO	no	7.40	835,194	30,834	5,184,070
	yes	3.67	783,024	29,768	1,432,892
GALEN	no	204.70	67,524,538	25,660	82,962
	yes	204.54	68,032,698	25,722	43,043

Table 2: test KB results

where PConcepts is the number of primitive concepts, nCD the number of completely defined concepts, NConcepts the number of non-primitive concepts, and Synonyms the number of synonyms. All experiments used v.0.99.4 of FaCT++ running under Linux on an Athlon 1.3GHz machine with 768Mb of memory.

The results of the classification tests are given in Table 2, where time is the time taken to classify the KB (in seconds), nOps is the number of expansion rule applications during the classification process, nTests is the number of subsumption tests, and nCache is the number of subsumptions that were computed using cached models [6].

Using CD speeds up the classification of NCI by a factor of more than 20. In both cases, all subsumption tests are solved cheaply using cached models, but more than ten million tests are performed when CD is not used; employing CD reduces this number to less than one million. Classification of GO is twice as fast with CD than without it. Again, GO has a simple structure, but is very broad, so CD still gives a significant reduction in the large number of cache based tests. GALEN behaves differently. It is the only KB where more "real" (non cache based) subsumption tests are performed with CD than without. This is due to the large number of non-primitive concepts that are classified in the middle of the hierarchy. Even in this case, however, saving large numbers of cache based tests leads to a slightly smaller overall classification time.

5 Discussion

The proposed classification technique is applicable to a large number of real-life ontologies, i.e., those where there are no non-absorbed GCIs. The best results are, of course, for ontologies with large numbers of primitive concepts and a simple structure, but even in cases where it has little beneficial effect, it does not appear to have any detrimental one. The number of such ontologies may decrease, because newly created ontologies will (probably) use more of

the expressive possibilities provided by modern DLs. With legacy ontologies, however, the method may prove to be very useful.

In [5] the so-called *bucket* method was proposed as a way to deal with broad and shallow hierarchies. In this method, when some concept in the hierarchy is found to have a large number of children, a new "virtual" concept is added to the hierarchy; this non-primitive concept is defined to be equivalent to the disjunction of some of the children of the original concept, and is used in fast cache-based comparison.

It is possible to use the bucket method at the second stage of the CD classification algorithm. However, this method will not improve first stage of the algorithm, since no search is actually performed there.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-ref/`.

[4] GOA project. European Bioinformatics Institute. `http://www.ebi.ac.uk/GOA/`.

[5] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[6] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[7] J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.

[8] D. Tsarkov and I. Horrocks. Efficient reasoning with range and domain constraints. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 41–50, 2004.