Exploiting Partial Information in Taxonomy Construction

Rob Shearer and Ian Horrocks

Oxford University Computing Laboratory, Oxford, UK

Abstract. One of the core services provided by OWL reasoners is *classification*: the discovery of all subclass relationships between class names occurring in an ontology. Discovering these relations can be computationally expensive, particularly if individual subsumption tests are costly or if the number of class names is large. We present a classification algorithm which exploits partial information about subclass relationships to reduce both the number of individual tests and the cost of working with large ontologies. We also describe techniques for extracting such partial information from existing reasoners. Empirical results from a prototypical implementation demonstrate substantial performance improvements compared to existing algorithms and implementations.

1 Introduction

Among the most important relationships between the class names occurring in an ontology is *subclassing*: a class B is a subclass of a class A if and only if every member of B must also be a member of A. One of the core services provided by reasoners for ontology languages is *classification*, the discovery of all such subclass relationships. In fact, classification is the primary (and often the only) reasoning service exposed by ontology engineering tools. The Protégé-OWL editor, for example, includes a "Reasoning" button which performs classification. The resulting hierarchy of subclass relationships is used to organize classes within all aspects of Protégé's interface, and the subclass relationships which arise as implicit consequences of an ontology are the primary mechanism authors use to check that the axioms they write are consistent with their intuitions about the structure of the domain. Finally, other reasoning services, such as explanation and query answering, typically exploit a cached version of the classification results; classification is thus usually the first task performed by a reasoner.

For some less expressive ontology languages, such as the OWL 2 EL profile,[1] it may be possible to derive all subclass relationships in a single computation [Baader *et al.*, 2005]. For OWL (2) DL, however, it is in general necessary to "deduce" the set of all such relationships by performing a number of *subsumption tests*; each such test checks whether or not there is a subclass relationship between two particular ontology classes.

In this paper, we present a novel algorithm which can greatly reduce the number of subsumption tests needed to classify a set of ontology classes. Our algorithm is also able to exploit *partial information* about subclass relationships—for example, the set of subclass relationships which are explicitly stated in the ontology—to further reduce the

[1] `http://www.w3.org/TR/owl2-profiles/`

number of tests; our approach thus generalizes a wide range of commonly-implemented optimizations. We further describe how a great deal of such partial information can be gathered from the data generated by reasoning systems when performing individual subsumption or consistency tests. Finally, we present an empirical evaluation that demonstrates the significant advantages of our techniques over the approaches implemented in existing reasoners, such as Pellet and FaCT++.

2 Overview

For a set of n classes, there are a total of n^2 possible subclass relationships, thus a naïve representation of classification results can require large amounts of storage. Most systems exploit the transitivity of subclassing and store only the "most specific" superclasses and "most general" subclasses of each class; we call such a compressed representation a *taxonomy*. A taxonomy lends itself to representation as a graph or tree, often called a *class hierarchy*.

It is clear that a set of n class names can be classified by simply performing all n^2 individual subsumption tests, but for the tree-shaped class hierarchies typically found in realistic ontologies much better results can be achieved using algorithms that construct the taxonomy incrementally. Class names are inserted into the taxonomy one by one, and the correct location for each is found by traversing the partially-constructed hierarchy, performing subsumption tests as each node of the graph is visited.

This kind of algorithm suffers from two main difficulties. First, individual subsumption tests can be computationally expensive—for some complex ontologies, even state-of-the-art reasoners may take a long time to perform a single test. Second, even when subsumption tests themselves are cheap, an ontology containing a very large number of class names will obviously result in a very large taxonomy, and repeatedly traversing this structure can be costly. This latter problem is particularly acute for the relatively flat (i.e., broad and shallow) tree-shaped hierarchies often found in large biomedical ontologies. In general, subsumption tests must be performed between every pair of class names with the same direct superclass(es), thus broad hierarchies require many such tests. These two difficulties clearly interact: large numbers of class names require large numbers of subsumption tests, each of which can be expensive.

The first difficulty is usually addressed by using an optimized construction that tries to minimize the number of subsumption tests performed in order to construct the taxonomy. Most implemented systems use an "enhanced traversal" algorithm due to Ellis [1991] and to Baader *et al.* [1994] which adds class names to the taxonomy one at a time using a two-phase strategy. In the first phase, the most specific superclasses of a class C are found using a top-down breadth-first search of the partially-constructed taxonomy. In this phase, subtrees of non-superclasses of C are not traversed, which significantly reduces the number of tests performed. The second phase finds the most general subclasses of C using a bottom-up search in a similar way. The algorithm exploits the structure of the ontology to identify "obvious" superclasses (so-called *told-subsumers*) of each class, and uses this information in a heuristic that chooses the order in which classes are added, the goal being to construct the taxonomy top-down; it also exploits information from the top-down search in order to prune the bottom-up search.[2]

The second difficulty can be addressed by optimizations that try to identify a subset of the class names for which complete information about the subclass relationships can be deduced without performing any individual subsumption tests. This can be achieved, e.g., by identifying *completely-defined* classes [Tsarkov *et al.*, 2007]—those between which only structurally-obvious subclass relationships hold. Having constructed part of the taxonomy using such a technique, the remaining class names can be added using the standard enhanced traversal algorithm.

In Section 3 we present a new classification algorithm that generalizes and refines the above techniques. Our approach is based on maintaining, and incrementally extending, two sorts of information: a set of pairs of classes A and B such that we know that A is a subclass of B (the *known subsumptions*), and a set of pairs of classes such that we know that A is not a subclass of B (the *known non-subsumptions*). The key insight is that information from these two sets can often be combined to derive new information. For example, if we know that A is a subclass of B, and that A is *not* a subclass of C, then we can conclude that B is not a subclass of C. In section 3.1 we show how to derive the largest possible set of such inferences.

Our classification algorithm is straightforward: at each stage we pick a pair of classes A and B which does not appear in either the set of known subsumptions or the set of known non-subsumptions, perform a subsumption test between the two, and use the result of the test to further extend the sets of known subsumptions and non-subsumptions. Each such test adds at least one pair to one of the sets (i.e. A and B themselves). Eventually, every possible pair of classes is present in one set or the other, and the set of known subsumptions thus contains all subsumptions between class names.

An important advantage of our algorithm is that the initial sets of known (non-)subsumptions may be empty, may include partial information about all classes, and/or may include complete information about some classes; in all cases the information is maximally exploited. Such information can be derived from a variety of sources, ranging from syntactic analysis of the ontology to existing classification results for a subset of class names from the ontology (e.g. independent modules or classified sub- or supersets of the ontology) to data derived in the course of reasoning.

In Section 4 we show how the models constructed by (hyper)tableau-based reasoners in the course of subsumption and satisfiability testing can be used as sources of partial information about subclass relationships. For example, if a reasoner produces a model containing an individual which is a member of both the class A and the complement (negation) of the class B, then A is clearly not a subclass of B; more sophisticated analysis based on the *dependency tracking* structures typically maintained by tableau reasoners also allows detection of subclass relationships. The models generated by tableau reasoners are typically very rich sources for this type of information; in fact, for ontologies which do not result in nondeterminism, including all OWL 2 EL ontologies, the model constructed by a hypertableau-based reasoner when checking the

[2] Other optimizations can be used to decrease the cost of individual subsumption tests (see, e.g., [Tsarkov *et al.*, 2007]), but these techniques are largely orthogonal to classification optimizations.

satisfiability of a class A will contain sufficient information to determine if A is (not) a subclass of B for all class names B occurring in the ontology.

Our approach to partial-information derivation provides an efficient generalization of the told-subsumer and completely-defined optimizations, both of which derive partial information from structural analysis of the ontology. When the known (non-)subsumption information is incomplete, our algorithm incrementally computes additional (non-)subsumption relationships, and maximally exploits the resulting information to refine the sets of known and possible subsumers; this can be seen as a generalization of the search-pruning optimizations introduced by Baader *et al.*.

We have used a prototypical implementation of our new algorithm to compare its behavior with that of the classification algorithms implemented in state-of-the-art OWL reasoners. The comparison shows that our algorithm can dramatically reduce the number of subsumption tests performed when classifying an ontology. Moreover, in contrast to the completely-defined optimization, the behavior of our algorithm degrades gracefully as the gap between the sets of initially-known and possible subsumption relationships increases.

3 Deducing a Quasi-Ordering

We first introduce some notation and definitions that will be useful in what follows.

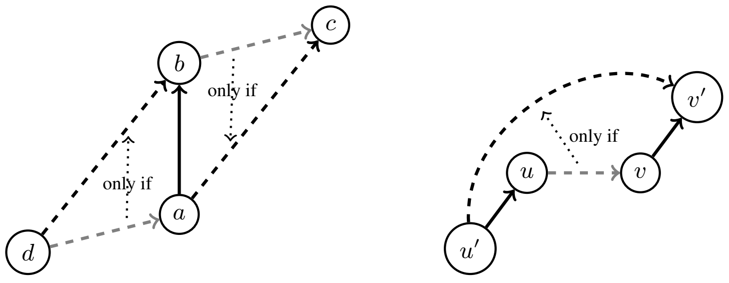
Given a set of elements $U = \{a, b, c, ...\}$, let R be a binary relation over U, i.e., a subset of $U \times U$. We say that there is a *path* from a to b in R if there exist elements $c_0, ..., c_n \in U$ such that $c_0 = a$, $c_n = b$, and $\langle c_i, c_{i+1} \rangle \in R$ for all $0 \leq i < n$. The *transitive closure* of R is the relation R^+ such that $\langle a, b \rangle \in R^+$ iff there is a path from a to b in R. The *transitive-reflexive closure* R^* of R is the transitive closure of the reflexive extension of R, i.e. $R^+ \cup \{\langle a, a \rangle \mid a \in U\}$.

A binary relation is a *quasi-ordering* if it is both reflexive and transitive. Clearly, the subsumption relation on a set of classes is a quasi-ordering. Note, however, that it is not a *partial-ordering*, because it is not antisymmetric: $C \sqsubseteq D$ and $D \sqsubseteq C$ does not imply that $C = D$.

The *restriction* of a relation R to a subset D of U is the relation $R[D] = R \cap (D \times D)$. All restrictions of a reflexive relation are reflexive, and all restrictions of a transitive relation are transitive; thus, a restriction of a quasi-ordering is itself a quasi-ordering. Further, if $R \subseteq S$ for relations R and S, then $R[D] \subseteq S[D]$ for all $D \subseteq U$.

Given a universe U, a quasi-ordering R over U and a finite set of elements $D \subseteq U$, we consider the problem of computing the restriction $R[D]$ via tests of the form $\langle a, b \rangle \in^? R$. If U is the set of (arbitrary) class expressions which can be represented in ontology language $\mathcal{L}$, R is the subsumption relation over U, and D is the set of class names occurring in an ontology $\mathcal{O}$ written in language $\mathcal{L}$, then computing $R[D]$ is equivalent to classifying $\mathcal{O}$, and the relevant tests are subsumption tests.

We assume that we begin with partial information about R: we are provided with a set $K = \{\langle a_0, b_0 \rangle, ..., \langle a_m, b_m \rangle\}$ where $\langle a_i, b_i \rangle \in R$ for $0 \leq i \leq m$, and also with a set $K_{neg} = \{\langle c_0, d_0 \rangle, ..., \langle c_n, d_n \rangle\}$ where $\langle c_i, d_i \rangle \notin R$ for $0 \leq i \leq n$. We call the set K the *known* portion of R. In this paper we do not operate on the set K_{neg} directly; our presentation instead refers to its complement $U \times U \setminus K_{neg}$, which we denote by



(a) Simple cases (b) General case

Fig. 1: Eliminating possible edges: if the solid edges are known to be in quasi-ordering R, then the gray edges can be in R only if the indicated dashed edges are in R.

P and call the *possible* portion of R. It is thus the case that $K \subseteq R \subseteq P$. If no partial information is available, then $K = \emptyset$ and $P = U \times U$.

We can use the result of each test $\langle a, b \rangle \in^? R$ to further refine the bounds on R by either adding $\langle a, b \rangle$ to K or removing it from P; eventually $K[D] = R[D] = P[D]$. We next show, however, that the bounds on R can sometimes be refined without performing additional tests by combining information from K and P.

3.1 Maximizing Partial Information

The key to minimizing the number of explicit tests required to discover $R[D]$ is maximizing the information gained from K and P. To do so, we exploit the knowledge that R is a quasi-ordering. In this case, $K \subseteq R$ obviously implies that $K^* \subseteq R$, so we can use K^* to obtain a tighter lower bound on R. Less obvious is the fact that we can also obtain a tighter upper bound on R by identifying tuples in P which are not consistent with K and the transitivity of R.

For example, consider the case shown in Figure 1(a). If we know that b is a successor of a in R (i.e., $\langle a, b \rangle \in K$), then an element c can be a successor of b only if it is also a successor of a (if $\langle a, c \rangle \notin P$ then $\langle b, c \rangle \notin R$). Further, a can be a successor of an element d only if b is also a successor of d.

Both of these examples are special cases of the structure shown in Figure 1(b): if u is a successor of u' and v' is a successor of v, then an edge from u to v would form a path all the way from u' to v', requiring v' to be a successor of u'. Since R is reflexive we can choose $u' = u$ or $v = v'$ to see that v can be a successor of u only if v is a successor of u' and v' is also a successor of u. We use this to formalize a subset $\lfloor P \rfloor_K$ of P, and show that $\lfloor P \rfloor_K$ is the tightest possible upper bound on R.

Definition 1 *Let K and P denote two relations such that $K^* \subseteq P$. We define the* reduction $\lfloor P \rfloor_K$ of P due to K *as follows:*

$$\lfloor P \rfloor_K = P \cap \{\langle u, v \rangle \mid \forall u', v' : \{\langle u', u \rangle, \langle v, v' \rangle\} \subseteq K^* \rightarrow \langle u', v' \rangle \in P\}$$

Lemma 1 *Let K and P denote two relations such that $K^* \subseteq P$. (i) For all quasi-orders R such that $K \subseteq R \subseteq P$, it is the case that $R \subseteq \lfloor P \rfloor_K$. (ii) Let S be a proper* *subrelation of $\lfloor P \rfloor_K$. Then there exists a quasi-ordering R such that $K \subseteq R \subseteq P$ and $R \not\subseteq S$; i.e. $\lfloor P \rfloor_K$ is minimal.*

Proof. *(i)* Let $\langle u, v \rangle$ be a tuple in R. For every u', v' such that $\{\langle u', u \rangle, \langle v, v' \rangle\} \subseteq K^*$, $K^* \subseteq R$ implies that $\{\langle u', u \rangle, \langle v, v' \rangle\} \subseteq R$. Because R is transitive and $\langle u, v \rangle \in R$, it must also be the case that $\langle u', v' \rangle \in R$ and thus that $\langle u', v' \rangle \in P$. Consequently, $\langle u, v \rangle \in \lfloor P \rfloor_K$, so $R \subseteq \lfloor P \rfloor_K$.

(ii) Choose elements a and b such that $\langle a, b \rangle \in \lfloor P \rfloor_K$ but $\langle a, b \rangle \notin S$. Let R be the transitive-reflexive closure of the relation $K \cup \{\langle a, b \rangle\}$. Clearly $K \subseteq R$ and $R \not\subseteq S$. Let $\langle u, v \rangle$ be any tuple in R. There are three cases:

1. $\langle u, v \rangle = \langle a, b \rangle$. Then $\langle u, v \rangle \in P$ since $\langle a, b \rangle \in \lfloor P \rfloor_K$ and $\lfloor P \rfloor_K \in P$.
2. $\langle u, v \rangle \in K^+$. Then $\langle u, v \rangle \in P$ since $K^* \subseteq P$.
3. $\langle u, a \rangle \in K^*$ and $\langle b, v \rangle \in K^+$. Then $\langle u, v \rangle \in P$ since $\langle a, b \rangle \in \lfloor P \rfloor_K$.

For any tuple $\langle u, v \rangle \in R$, it is the case that $\langle u, v \rangle \in P$, thus $K \subseteq R \subseteq P$ and $R \not\subseteq S$.
□

Note that $\lfloor P \rfloor_K$ itself is not necessarily transitive: given three elements a, b, and c and the relation $P = \{\langle a, a \rangle, \langle b, b \rangle, \langle c, c \rangle, \langle a, b \rangle, \langle b, c \rangle\}$, it is the case that $\lfloor P \rfloor_\emptyset = P$. Of course no transitive subrelation R of P contains both $\langle a, b \rangle$ and $\langle b, c \rangle$.

3.2 Taxonomy Construction and Searching

As described in Section 3.1, given relations K and P such that $K \subseteq R \subseteq P$ for some unknown quasi-ordering R, a tuple $\langle a, b \rangle$ is an element of R if $\langle a, b \rangle \in K^*$, and $\langle a, b \rangle$ is not an element of R if $\langle a, b \rangle \notin \lfloor P \rfloor_K$; the only "unknown" elements of R are the tuples in $\lfloor P \rfloor_K \setminus K^*$. Further, if $\langle a, b \rangle \in \lfloor P \rfloor_K \setminus K^*$, then a test of the form $\langle a, b \rangle \in^? R$ provides additional information which can be used to extend K or restrict P. This suggests the following simple procedure for deducing the restriction $R[D]$ of a quasi-ordering R to domain D:

```
COMPUTE-ORDERING($K, P, D$)
1  while $K^*[D] \neq \lfloor P \rfloor_K[D]$
2      do choose some $a, b \in D$ such that $\langle a, b \rangle \in \lfloor P \rfloor_K \setminus K^*$
3         if $\langle a, b \rangle \in^? R$ then add $\langle a, b \rangle$ to $K$
4                                  else remove $\langle a, b \rangle$ from $P$
5  return $K[D]$
```

Completely recomputing K^* and $\lfloor P \rfloor_K$ in each iteration of the above loop is clearly inefficient. Practical implementations would instead maintain both relations and update them as P and K change. Techniques for updating the transitive-reflexive closure of a relation are well-known; we provide below a naïve algorithm that, given $\lfloor P \rfloor_K$, $K' \supseteq K$ and $P' \subseteq P$, computes an updated relation $\lfloor P' \rfloor_{K'}$.

The algorithm exploits the technique for eliminating edges that was described in Section 3.1 and Figure 1(b): it removes a tuple $\langle u, v \rangle$ from the set of possible tuples $\lfloor P \rfloor_K$ when adding it to the set of known tuples K' would imply, due to the transitivity of R, that some other tuple would be at the same time both known (i.e., in K') and not possible (i.e., not in $\lfloor P' \rfloor_{K'}$). This is done incrementally by considering tuples that have either become known (i.e., are in $K' \setminus K$) or been shown to be impossible (i.e., are in $\lfloor P \rfloor_K \setminus P'$).

First, $\lfloor P \rfloor_K$ is copied to $\lfloor P' \rfloor_{K'}$. Then, in lines 2–4, each tuple $\langle u', v' \rangle$ that has been shown to be impossible is considered and, if there are tuples $\langle u', u \rangle$ and $\langle v, v' \rangle$ in K', then $\langle u, v \rangle$ is clearly not possible either (it would imply that $\langle u', v' \rangle$ is not only possible but known) and so is removed from $\lfloor P \rfloor_K$. Next, in lines 5–13, each tuple $\langle x, y \rangle$ that has become known is considered. There are two possible cases: one where x, y correspond to u', u in Figure 1(b), and one where they correspond to v, v'. Lines 6–9 deal with the first case: if there are tuples $\langle u, v \rangle$ in $\lfloor P \rfloor_K$ and $\langle v, v' \rangle$ in K', but $\langle u', v' \rangle$ is not in P', then $\langle u, v \rangle$ is clearly not possible (it would imply that $\langle u', v' \rangle$ is not only possible but known) and so is removed from $\lfloor P \rfloor_K$. Lines 10–13 deal similarly with the second case: if there are tuples $\langle u, v \rangle$ in $\lfloor P \rfloor_K$ and $\langle u', u \rangle$ in K', but $\langle u', v' \rangle$ is not in P', then $\langle u, v \rangle$ is removed from $\lfloor P \rfloor_K$.

```
PRUNE-POSSIBLES($\lfloor P \rfloor_K, K, P', K'$)
 1  $\lfloor P' \rfloor_{K'} \leftarrow \lfloor P \rfloor_K$
 2  for each $\langle u', v' \rangle \in \lfloor P \rfloor_K \setminus P'$
 3        do for each $u, v$ such that $\langle u', u \rangle \in K'$ and $\langle v, v' \rangle \in K'$
 4              do remove $\langle u, v \rangle$ from $\lfloor P' \rfloor_{K'}$
 5  for each $\langle x, y \rangle \in K' \setminus K$
 6        do let $u' \leftarrow x$ and $u \leftarrow y$
 7           for each $v$ such that $\langle u, v \rangle \in \lfloor P \rfloor_K$
 8              do if there exists $v'$ such that $\langle v, v' \rangle \in K'$ and $\langle u', v' \rangle \notin P'$
 9                    then remove $\langle u, v \rangle$ from $\lfloor P' \rfloor_{K'}$
10           let $v \leftarrow x$ and $v' \leftarrow y$
11              do for each $u$ such that $\langle u, v \rangle \in \lfloor P \rfloor_K$
12                    do if there exists $u'$ such that $\langle u', u \rangle \in K'$ and $\langle u', v' \rangle \notin P'$
13                          then remove $\langle u, v \rangle$ from $\lfloor P' \rfloor_{K'}$
14           return $\lfloor P' \rfloor_{K'}$
```

In the case where no information about the quasi-ordering $R[D]$ is available other than K and P, the COMPUTE-ORDERING procedure performs well. In many cases, however, some general properties of $R[D]$ can be assumed. In the case where R represents subsumption relationships between class expressions, for example, $R[D]$ is typically much smaller than $D \times D$ (i.e., relatively few pairs of class names are in a subsumption relationship). In such cases, it is beneficial to use heuristics that exploit the (assumed) properties of $R[D]$ when choosing a and b in line 2 of the above procedure.

We summarize below a variant of COMPUTE-ORDERING which performs well when the restriction to be computed is treelike in structure and little information about the ordering is available in advance. This procedure is designed to perform individual tests in an order similar to the enhanced traversal algorithm; however, it minimizes the number of individual tests performed by maximally exploiting partial information.

The algorithm chooses an element of $a \in D$ for which complete information about $R[D]$ is not yet known. It identifies the subset $V^{\uparrow} \subseteq D$ of elements b for which $\langle a, b\rangle \in R$, and the subset $V^{\downarrow} \subseteq D$ of elements b for which $\langle b, a\rangle \in R$, updating K and P accordingly. In order to compute these sets efficiently, we make use of the subroutines SUCCESSORS and PREDECESSORS, which perform the actual tests. The SUCCESSORS and PREDECESSORS functions are derived from the enhanced traversal algorithm: they perform a breadth-first search of the *transitive reduction* $K^{\diamond}$ of the known subsumptions K—the smallest relation whose transitive closure is K^*. In order to avoid the cost of repeated traversals of $K^{\diamond}$, we restrict the searches to, respectively, the possible successors and predecessors of a. We omit the details of these search routines for the sake of brevity.

```
COMPUTE-ORDERING-2($K, P, D$)
 1  while $K^*[D] \neq \lfloor P \rfloor_K[D]$
 2      do choose some $a, x \in D$ s.t. $\langle a, x\rangle \in \lfloor P \rfloor_K \setminus K^*$ or $\langle x, a\rangle \in \lfloor P \rfloor_K \setminus K^*$
 3          let $B$ be the possible successors of $a$, i.e. $D \cap \{b \mid \langle a, b\rangle \in \lfloor P \rfloor_K \setminus K^*\}$
 4          if $B \neq \emptyset$ then $V^{\uparrow} \leftarrow$ SUCCESSORS($a, K^{\diamond}[B]$)
 5                      add $\langle a, b\rangle$ to $K$ for every element $b$ of $V^{\uparrow}$
 6                      remove $\langle a, b\rangle$ from $P$ for every element $b$ of $B \setminus V^{\uparrow}$
 7          let $B$ be the possible predecessors of $a$, i.e. $D \cap \{b \mid \langle b, a\rangle \in \lfloor P \rfloor_K \setminus K^*\}$
 8          if $B \neq \emptyset$ then $V^{\downarrow} \leftarrow$ PREDECESSORS($a, K^{\diamond}[B]$)
 9                      add $\langle b, a\rangle$ to $K$ for every element $b$ of $V^{\downarrow}$
10                      remove $\langle b, a\rangle$ from $P$ for every element $b$ of $B \setminus V^{\downarrow}$
11  return $K[D]$
```

3.3 Example

Consider the process of using COMPUTE-ORDERING-2 to discover the subsumption relation $\{\langle a, a\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle c, a\rangle, \langle c, c\rangle, \langle d, a\rangle, \langle d, b\rangle, \langle d, d\rangle\}$ with no initial partial information available. We initialize K to $\{\langle a, a\rangle\langle b, b\rangle, \langle c, c\rangle, \langle d, d\rangle\}$ and P to $\{a, b, c, d\} \times \{a, b, c, d\}$; this situation is shown in the left diagram of Figure 2. Each element appears in a tuple occurring in $K^* \setminus \lfloor P \rfloor_K$, so on the first execution of line 2 of COMPUTE-ORDERING-2 we are free to choose any element; assume that we choose d. Then SUCCESSORS performs the tests $d \sqsubseteq^? a$, $d \sqsubseteq^? b$, and $d \sqsubseteq^? c$ (discovering that a and b are the only successors of d), we add $\langle a, d\rangle$ and $\langle b, d\rangle$ to K, and we remove $\langle c, d\rangle$ from P. PREDECESSORS performs the tests $a \sqsubseteq^? d$, $b \sqsubseteq^? d$, and $c \sqsubseteq^? d$ (discovering that d has no predecessors), and we remove each of $\langle a, d\rangle$, $\langle b, d\rangle$, and $\langle c, d\rangle$ from P. Further, because $d \sqsubseteq a$ but $d \not\sqsubseteq c$, we can conclude that $a \not\sqsubseteq c$; similar reasoning shows that $b \not\sqsubseteq d$; $\lfloor P \rfloor_K$ is thus restricted to the set $\{\langle a, a\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle b, b\rangle, \langle c, a\rangle, \langle c, b\rangle, \langle c, c\rangle, \langle d, a\rangle, \langle d, b\rangle, \langle d, d\rangle\}$. The states of K^* and $\lfloor P \rfloor_K$ at this point are shown in the left-center diagram of Figure 2.

In the next iteration through the COMPUTE-ORDERING-2 loop we cannot choose d since it does not occur in any tuple of $K^* \setminus \lfloor P \rfloor_K$; assume that we instead choose b. The only possible successor of b is a, so SUCCESSORS searches this one-element subgraph by performing the single test $b \sqsubseteq^? a$ (which returns TRUE), and we add $\langle b, a\rangle$ to K. The

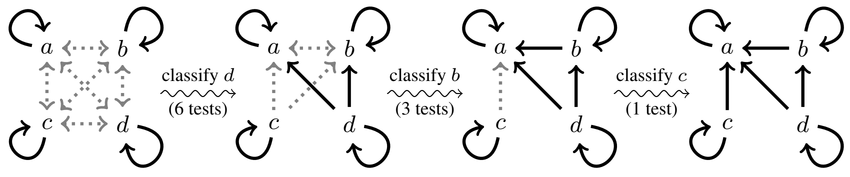


Fig. 2: As classification proceeds, known edges (denoted by solid black arrows) are discovered, and possible edges (denoted by dotted gray arrows) are eliminated.

element d is already known to be a predecessor of b, so PREDECESSORS searches the subgraph $K^{\diamond}[\{a, c\}]$ by performing the tests $a \sqsubseteq^{?} b$ and $c \sqsubseteq^{?} b$, finding no predecessors, and the two corresponding tuples are removed from P. The states of K^{*} and $\lfloor P \rfloor_K$ are shown in the right-center diagram of Figure 2.

After two iterations, the set $K^{*} \setminus \lfloor P \rfloor_K$ contains only the pair $\langle c, a \rangle$; if we choose c in the next iteration then SUCCESSORS tests $c \sqsubseteq^{?} a$, we add $\langle c, a \rangle$ to K, and $K^{*} = \lfloor P \rfloor_K$. The final subsumption relation is given by K^{*} and is shown in the right-hand diagram of Figure 2.

COMPUTE-ORDERING-2 thus classifies this ontology using 10 subsumption tests instead of the 16 required by a naïve brute-force approach. Note, however, that the results of the seven tests $a \sqsubseteq^{?} c$, $a \sqsubseteq^{?} d$, $b \sqsubseteq^{?} a$, $b \sqsubseteq^{?} d$, $c \sqsubseteq^{?} a$, $c \sqsubseteq^{?} b$, and $d \sqsubseteq^{?} b$ are sufficient to extend K and restrict P such that $K^{*} = \lfloor P \rfloor_K$, providing a full classification for this ontology. Identifying such a minimal set of tests is, however, extremely difficult without prior knowledge of the final taxonomy.

4 Extracting Subsumption Information from Models

We next turn our attention to the specific case of identifying all subsumption relationships between the class names occurring in an ontology $\mathcal{O}$. Instead of treating a reasoning service as an oracle that answers boolean queries of the form "is A subsumed by B w.r.t. $\mathcal{O}$?" (which we will write $\mathcal{O} \models^{?} A \sqsubseteq B$), we consider how information generated internally by common reasoning algorithms can be exploited to discover information about the subsumption quasi-ordering.

4.1 Identifying Non-Subsumptions

Most modern reasoners for Description Logics, including HermiT, Pellet, and FaCT++, transpose subsumption queries into satisfiability problems; in particular, to determine if $\mathcal{O} \models A \sqsubseteq \bot$, these reasoners test whether the class A is satisfiable w.r.t. $\mathcal{O}$. They do this by trying to construct (an abstraction of) a Tarski-style *model* of $\mathcal{O}$ in which the extension of A is nonempty. We begin by providing an abbreviated formalization of such models (see Baader *et al.* [2003] for more details):

Definition 2 *Given sets of class names N_C, property names N_R and individual names N_I, an* interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a nonempty set $\Delta^{\mathcal{I}}$ and an* interpretation function $\cdot^{\mathcal{I}}$ *which maps every element of N_C to a subset of $\Delta^{\mathcal{I}}$, every element of N_R to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and every element of N_I to an element of $\Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is a model of an axiom $A \sqsubseteq B$ if $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ (similar definitions hold for other kinds of statement); it is a model of an ontology $\mathcal{O}$ if it models every statement in $\mathcal{O}$.*

Let A and B be classes. A model $\mathcal{I}$ of $\mathcal{O}$ is a witness for the satisfiability of A w.r.t. $\mathcal{O}$ *if $A^{\mathcal{I}}$ is nonempty; it is a* witness for the non-subsumption $A \not\sqsubseteq B$ w.r.t. $\mathcal{O}$ *if $A^{\mathcal{I}} \not\subseteq B^{\mathcal{I}}$, i.e., if there exists $i \in \Delta^{\mathcal{I}}$ s.t. $i \in A^{\mathcal{I}}$ and $i \notin B^{\mathcal{I}}$.*

The algorithms in question typically represent the model being constructed as an ABox, i.e., as a set of *assertions* of the form $x : C$ and $\langle x, y \rangle : R$ for individuals x, y, classes C, and properties R [Baader *et al.*, 2003]. An ABox including the assertion $x : C$ represents a model in which $x^{\mathcal{I}} \in C^{\mathcal{I}}$. To construct a witness for the satisfiability of a class A, the ABox is initialised with an assertion $x : A$ and the construction proceeds in a goal-directed manner by adding further assertions only as necessary in order to ensure that the ABox represents a model of $\mathcal{O}$.

Assuming that the construction is successful, the resulting ABox/model provides a rich source of information. For example, for any class B such that $x : (\neg B)$ is in the ABox, it is the case that $x^{\mathcal{I}} \notin B^{\mathcal{I}}$; thus the model is a witness for the non-subsumption $\mathcal{O} \models A \not\sqsubseteq B$ for all such classes B. In many cases, the non-presence of $x : B$ in the ABox is sufficient to conclude the relevant non-subsumption; in fact, when using a hypertableau algorithm, this is *always* the case. When using a tableau algorithm, the non-subsumption conclusion can be drawn if B is a so-called primitive concept, i.e., if the unfoldable part of the TBox includes an axiom $B \sqsubseteq C$ for some concept C; in this case, adding $x : (\neg B)$ to the ABox trivially results in a witness for the non-subsumption (this is closely related to the widely employed model merging optimization [Tsarkov *et al.*, 2007]).

The goal-directed nature of the ABox construction means that the models constructed are typically quite small. As a result, these models tend to be extremely rich in non-subsumption information: in a typical witness for the satisfiability of A, i.e., a model $\mathcal{I}$ of $\mathcal{O}$ with $i \in A^{\mathcal{I}}$, there will be relatively few other class names B such that $i \in B^{\mathcal{I}}$, and thus $\mathcal{I}$ will identify the vast majority of class names in $\mathcal{K}$ as non-subsumers of A. For this reason, it is almost always more efficient to record the set $P_A = \{B \mid i \in A^{\mathcal{I}} \text{ and } i \in B^{\mathcal{I}} \text{ for some } i\}$ of "possible subsumers" of A.

4.2 Identifying Subsumptions

While single models allow us to detect non-subsumptions, additional information about the space of possible models is required in order to identify subsumption relationships. Sound and complete tableau reasoning algorithms systematically explore the space of all "canonical" models (typically tree- or forest-shaped models), on the basis that, if any model exists, then one of these canonical models also exists. In particular, when $\mathcal{O}$ includes disjunctions or other sources of nondeterminism, it may be necessary to choose between several possible ways of modeling such statements, and to backtrack and try other possible choices if the construction fails.

For such algorithms, it is usually easy to show that, if the ABox was initialized with $x : A$, the construction did not involve any nondeterministic choices, and the resulting ABox includes the assertion $x : B$, then it is the case that in any model $\mathcal{I}$ of $\mathcal{K}$, $i \in A^{\mathcal{I}}$ implies $i \in B^{\mathcal{I}}$, i.e., that $\mathcal{K} \models A \sqsubseteq B$. Moreover, as we have already seen in Section 4.1, such an ABox is (at least in the hypertableau case) a witness to the non-subsumption $\mathcal{K} \models A \not\sqsubseteq B$ for all class names B such that $x : B$ is not in the ABox. Thus, when testing the satisfiability of a class A, it may be possible to derive *complete information* about the subsumers of A.

The hypertableau-based HermiT reasoner is designed to reduce nondeterminism, and avoids it completely when dealing with Horn-$\mathcal{SHIQ}$ (and OWL 2 EL) ontologies; for such ontologies it is thus able to derive complete information about the subsumers of a class A using a single satisfiability test. This allows HermiT to derive all relevant subsumption relationships in a Horn-$\mathcal{SHIQ}$ ontology as a side effect of performing satisfiability tests on each of the class names [Motik *et al.*, 2007].

This idea can be extended so as to also derive useful information from nondeterministic constructions by exploiting the dependency labeling typically used to enable "dependency-directed backtracking"—an optimization which reduces the effects of nondeterminism in reasoning [Horrocks, 1997]. In the resulting ABoxes, each assertion is labelled with the set of choice points on which it depends. An empty label indicates that the relevant assertion will always be present in the ABox, regardless of any choices made during the construction process. Thus, if the ABox is initialized with $x : A$, an empty-labelled assertion $x : B$ in the resulting ABox can be treated in the same way as if the construction had been completely deterministic. Performing a satisfiability test on A may, therefore, allow some subsumers of A to be identified even when nondeterministic choices are made during reasoning. In practice, almost all of the actual subsumers of A can usually be identified in this way.

It is easy to see that this idea is closely related to, and largely generalizes, the told subsumer and completely-defined optimizations. For a completely defined class name A, a satisfiability test on A will be deterministic (and typically rather trivial), and so will provide complete information about the subsumers of A. Similarly, if B is a told subsumer of A, then an ABox initialized with $x : A$ will always produce $x : B$, and almost always deterministically. (It is theoretically possible that $x : B$ will be added first due to some nondeterministic axiom in the ontology).

5 Related Work

Computing a quasi- (or partial-) ordering for a set of n incomparable elements clearly requires n^2 individual tests—naïvely comparing all pairs is thus "optimal" by the simplest standard. The literature therefore focuses on a slightly more sophisticated metric which considers both the number of elements in the ordering as well as the *width* of the ordering—the maximum size of a set of mutually incomparable elements. Faigle and Turán [1985] have shown that the number of comparisons needed to deduce an ordering of n elements with width w is at most $O(wn \log(n/w))$ and Daskalakis *et al.* provide an algorithm which approaches this bound by executing $O(n(w + \log n))$ comparisons [2007]. Taxonomies, however, tend to resemble trees in structure, and the width of a subsumption ordering of n elements is generally close to $n/2$. Further, the algorithms of Faigle and Turán as well as Daskalakis *et al.* rely on data structures which require $O(nw)$ storage space even in the best case, and thus exhibit quadratic performance when constructing a taxonomy.

A taxonomy-construction strategy which performs well for tree-like relations is described by Ellis [1991]: elements are inserted into the taxonomy one at a time by finding, for each element, its subsumers using a breadth-first search of all previously-inserted elements top-down, and then its subsumees using a breadth-first search bottom-up. Baader *et al.* further refine this technique to avoid redundant subsumption tests during each search phase: during the top search phrase, a test $\mathcal{K} \models^? A \sqsubseteq B$ is performed only if $\mathcal{K} \models A \sqsubseteq C$ for all subsumers C of B [1994]. This can be seen as a special case of our $\lfloor P \rfloor_K$ pruning of possible subsumers, with the restriction that it only applies to subsumption tests performed in a prescribed order.

The traversal algorithms described by Ellis and by Baader *et al.* perform subsumption tests between every pair of siblings in the final taxonomy. Such algorithms are thus very inefficient for taxonomies containing nodes with large numbers of children; completely flat taxonomies result in a quadratic number of subsumption tests. Haarslev and Möller propose a way to avoid the inefficiencies of these traversals by *clustering* a group of siblings $A_1, ..., A_n$ by adding the class expression $\bigsqcap_{1 \leq i \leq n} A_i$ to the taxonomy [2001]. Our approach can easily incorporate this technique by including partial or complete information about such new class expressions in K and P. In practice, however, the partial information extracted from tableau models almost always includes non-subsumptions between siblings in such flat hierarchies, so these refinements are unnecessary.

Baader *et al.* also describe techniques for identifying subsumers without the need for multiple subsumption tests by analyzing the syntax of class definitions in an ontology: if an ontology contains an axiom of the form $A \sqsubseteq B \sqcap C$ where A and B are class names, then B is a "told subsumer" of A, as are all the told subsumers of B. The various simplification and absorption techniques described by Horrocks [1997] increase the applicability of such analysis. Haarslev *et al.* further extend this analysis to detect non-subsumption: an axiom of the form $A \sqsubseteq \neg B \sqcap C$ implies that A and B are disjoint, thus neither class name subsumes the other (unless both are unsatisfiable) [2001]. Tsarkov *et al.* describe a technique for precisely determining the subsumption relationships between "completely defined classes"—class names whose definitions contain only conjunctions of other completely defined classes [2007]. All these optimizations can be seen as special cases of (non-)subsumption information being derived from (possibly incomplete) calculi as described in Section 4.

6 Empirical Evaluation

In order to determine if our new algorithm is likely to improve classification performance in practice we conducted two experiments using large ontologies derived from life-science applications.

First, we compared the performance of our new algorithm with the enhanced traversal algorithm. In order to analyze how much improvement is due to the information

Table 1: Algorithm Comparison

Relation Size		ET		New	
Known	Possible	Tests	Seconds	Tests	Seconds
335 476	335 476	0	190	0	17
335 476	2 244 050	152 362	246	24 796	22
335 476	4 147 689	303 045	257	49 308	31
335 476	6 046 804	455 054	292	73 945	33
335 476	7 940 847	606 205	305	98 613	34
251 880	335 476	80 878	634	19 773	28
251 880	2 244 050	439 002	740	50 143	32
251 880	4 147 689	794 513	809	79 038	40
251 880	6 046 804	1 151 134	836	107 416	46
251 880	7 940 847	1 506 752	919	136 190	50
168 052	335 476	143 913	1079	62 153	62
168 052	2 244 050	673 768	1267	146 823	91
168 052	4 147 689	1 201 904	1320	226 670	93
168 052	6 046 804	1 729 553	1414	304 784	98
168 052	7 940 847	-	-	381 330	130

extracted directly from models and how much is due to our new approach to taxonomy construction, we extend the enhanced traversal algorithm such that it first performs a satisfiability test on every class name and constructs a cache of information derived from the resulting models using the techniques described in Section 4. During the subsequent taxonomy construction, subsumption tests are performed only if the relevant subsumption relationship cannot be determined by consulting the cache. Note that this caching technique strictly subsumes the "told subsumer" and "primitive component" optimizations described by Baader *et al.*.

We implemented both algorithms within the HermiT reasoner [Motik *et al.*, 2007] and performed testing using an OWL version of the well-known US National Cancer Institute thesaurus (NCI), a large but simple ontology containing 27,653 classes.[3] The models constructed by HermiT during satisfiability testing of these classes provide complete information about the subsumption ordering for this ontology, so both algorithms are able to classify it without performing any additional tests. To study how the algorithms compare when less-than-complete information is available, we limited the amount of information extracted from HermiT's models, reducing the number of known subsumptions and increasing the number of possible subsumptions to varying degrees. The number of full subsumption tests required for classification as well as the total running times (including both classification and satisfiability testing) for each implementation are given in Table 1.

As the table shows, our simple implementation of the enhanced traversal algorithm (ET) is substantially slower than the new algorithm even when complete information is available; this is the result of the "insertion sort" behavior of ET described in Section 5.

[3] The latest version of the NCI ontology contains over 34,000 classes; the older version used here was, however, sufficient for our purposes.

When complete information is not available, our algorithm consistently reduces the number of subsumption tests needed to fully classify the ontology by an order of magnitude.

In a second experiment, we compared the implementation of our new algorithm in HermiT with the widely-used Description Logic classifiers FaCT++ and Pellet. Both of these systems are quite mature and implement a wide range of optimizations to both taxonomy construction and subsumption reasoning; we were thus able to compare our new algorithm with existing state-of-the-art implementations.

In this case, in addition to NCI we used the Gene Ontology (GO), and the well-known GALEN ontology of medical terminology[4]. Both NCI and GO have been specifically constructed to fall within the language fragment which existing reasoners are able to classify quickly; GALEN, in contrast, necessitates substantially more difficult subsumption testing but contains an order of magnitude fewer class names. In order to estimate how the different algorithms would behave with more expressive ontologies, for each ontology $\mathcal{O}$ we constructed two extensions: $\mathcal{O}^{\exists}$, which adds the single axiom $\top \sqsubseteq \exists R.A$ for a fresh property name R and fresh class name A, and $\mathcal{O}^{\sqcup}$ which adds the axiom $\top \sqsubseteq A \sqcup B$ for fresh class names A and B. For NCI we constructed a further extension NCI$^{\exists\forall}$ by adding the axioms $\top \sqsubseteq \exists R.A$ and $C \sqsubseteq \forall R.B$ for each of the 17 most general class names C occurring in the ontology. Each of these extensions increases the complexity of individual subsumption tests and reduces the effectiveness of optimizations that try to avoid performing some or all of the tests that would otherwise be needed during classification.

The number of class names occurring in each ontology as well as the number of tests performed (including all class satisfiability and subsumption tests) and the total time taken by each reasoner to fully classify each ontology are shown in Table 2. The Pellet system makes use of a special-purpose reasoning procedure for ontologies that fall within the $\mathcal{EL}$ fragment [Baader *et al.*, 2005]; for such ontologies we do not, therefore, list the number of subsumption tests performed by Pellet.

As Table 2 shows, HermiT's new classification algorithm dramatically reduces the number of subsumption tests performed when classifying these ontologies. This does not, however, always result in faster performance. This is largely due to optimizations used by the other reasoners which greatly reduce the cost of subsumption testing for simple ontologies: the overwhelming majority of subsumption tests performed by FaCT++, for example, can be answered using the pseudo-model merging technique described by Horrocks [1997].

Most of these optimizations could equally well be used in HermiT, but in the existing implementation each subsumption test performed by HermiT is far more costly. The number of subsumption tests performed by HermiT is, however, far smaller than for the other reasoners, and its performance also degrades far more gracefully as the complexity of an ontology increases: adding a single GCI or disjunction to an ontology can prevent the application of special-case optimizations in Pellet and FaCT++, greatly increasing the cost of subsumption testing and, due to the very large number of tests being

[4] All test data is available from `http://www.comlab.ox.ac.uk/rob.shearer/2009/iswc-classification-ontologies.tgz`

Table 2: System Comparison

		FaCT++		Pellet		HermiT	
Ontology	Classes	Tests	Seconds	Tests	Seconds	Tests	Seconds
NCI	27 653	4 506 097	2.3	-	16.1	27 653	22
NCI$^{\exists}$	27 654	8 658 610	4.4	-	16.7	27 654	21.0
NCI$^{\sqcup}$	27 655	8 687 327	5.1	10 659 876	95.4	48 389	37.0
NCI$^{\exists\forall}$	27 656	18 198 060	473.9	10 746 921	1098.3	27 656	20.8
GO	19 529	26 322 937	8.6	-	6.0	19 529	9.2
GO$^{\exists}$	19 530	26 904 495	12.7	-	6.9	19 530	9.7
GO$^{\sqcup}$	19 531	26 926 653	15.5	21 280 377	170.0	32 614	15.2
GALEN	2749	313 627	11.1	131 125	8.4	2749	3.3
GALEN$^{\exists}$	2750	327 756	473.5	170 244	9.7	2750	3.5
GALEN$^{\sqcup}$	2751	329 394	450.5	175 859	9.8	4657	40.5

performed, vastly increasing the time required for classification. The NCI$^{\exists\forall}$ ontology, for example, eliminates any benefit from the pseudo-model merging optimization (since no two pseudo-models can be trivially merged), and this causes the classification time to increase by roughly two orders of magnitude for both Pellet and FaCT++. In contrast, HermiT's classification time is unaffected. The relatively poor performance of HermiT on the GALEN$^{\sqcup}$ ontology is due to the fact that the underlying satisfiability testing procedure is particularly costly when there are large numbers of branching points, even if no backtracking is actually required.

7 Discussion and Future Work

We have described a new algorithm for taxonomy construction that effectively exploits partial information derived from structural analysis and/or reasoning procedures, and we have shown that, when compared to the widely-used enhanced traversal algorithm, it can dramatically reduce both the number of individual comparisons and the total processing time for realistic data sets. For simple ontologies, our prototype implementation makes the HermiT reasoner competitive with state-of-the-art reasoners which implement special-purpose optimizations of the subsumption testing procedure for such cases; on more expressive ontologies our new system substantially outperforms existing systems.

Future work will include extending HermiT to incorporate some of the subsumption testing optimizations used in other systems, in particular reducing the overhead cost of individual subsumption tests. We believe that this will greatly improve HermiT's performance on simple ontologies; as we have seen, it is already highly competitive on more complex ontologies.

The procedure we describe in Section 4 extracts subsumption relationships involving only the class used to initialize a model. This is because the dependency labeling implemented in tableau reasoners is currently designed only to allow the application of dependency-directed backtracking, and discards a great deal of dependency information. We intend to explore more sophisticated dependency labeling strategies which allow the extraction of additional subsumption information.

We also want to investigate meaningful complexity bounds for taxonomy searching and construction tasks. As we have seen, a completely naïve search routine is optimal if only the number of elements is considered. We will attempt to obtain tighter bounds for certain classes of relation: relations with linear taxonomy graphs, for example, can be deduced with only $n \log n$ comparisons. Bounds based on more sophisticated metrics may also be possible; e.g., bounds based on the size of the subsumption relation instead of the number of elements.

Finally, preliminary testing demonstrates that when significant partial information is available, the COMPUTE-ORDERING-2 procedure, based on the breadth-first search of the enhanced traversal algorithm, offers little advantage over COMPUTE-ORDERING, which performs tests in an arbitrary order; in many cases the performance of COMPUTE-ORDERING-2 is actually worse. Investigating other heuristics for choosing the order in which to perform tests will also be part of our future work.

References

1994. Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans jurgen Pro Tlich, and Enrico Franconi. 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*, 4:270–281, 1994.

2003. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

2005. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 364–369, 2005.

2007. Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha Riesenfeld, and Elad Verbin. Sorting and selection in posets. *CoRR*, abs/0707.1532, 2007.

1991. Gerard Ellis. Compiled hierarchical retrieval. In *In 6th Annual Conceptual Graphs Workshop*, pages 285–310, 1991.

1985. Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. In Kurt Mehlhorn, editor, *STACS*, volume 182 of *Lecture Notes in Computer Science*, pages 109–118. Springer, 1985.

2001. V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nebel, editor, *Proceedings of Seventeenth International JointŁ Conference on Artificial Intelligence, IJCAI-01*, pages 161–166, 2001.

2001. Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting pseudo models for tbox and abox reasoning in expressive description logics. In *IJCAR*, pages 61–75, 2001.

1997. Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

2007. Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In Frank Pfenning, editor, *Proc. of the 21st Conference on Automated Deduction (CADE-21)*, volume 4603 of *LNAI*, pages 67–83, Bremen, Germany, July 17–20 2007. Springer.

2007. Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimising terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 2007.