An Empirical Analysis of Optimization Techniques for Terminological Representation Systems

or: Making KRIS get a move on*

Franz Baader, **Bernhard Hollunder**,
Bernhard Nebel, **Hans-Jürgen Profitlich**
German Research Center for AI (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
e-mail: ⟨*last name*⟩@dfki.uni-sb.de

and

Enrico Franconi
Istituto per la Ricerca Scientifica e Tecnologica (IRST)
38050 Povo TN, Italy
e-mail: franconi@irst.it

Abstract

We consider different methods of optimizing the classification process of terminological representation systems, and evaluate their effect on three different types of test data. Though these techniques can probably be found in many existing systems, until now there has been no coherent description of these techniques and their impact on the performance of a system. One goal of this paper is to make such a description available for future implementors of terminological systems. Building the optimizations that came off best into the KRIS system greatly enhanced its efficiency.

*This is a revised and extended version of a paper presented at the *3rd International Conference on Principles of Knowledge Representation and Reasoning, October 1992, Cambridge, MA.*

1 Introduction

Terminological representation systems can be used to represent the taxonomic and conceptual knowledge of a problem domain in a structured way. To describe this kind of knowledge, one starts with atomic concepts (unary predicates) and roles (binary predicates), and defines more complex concepts and roles using the operations provided by the concept language of the particular formalism. In addition to this concept description formalism, most terminological representation systems also have an assertional component, which can be used to express facts about objects in the application domain.

Of course, it is not enough to have a system that just stores concept definitions and assertional facts. The system must also be able to reason about this knowledge. An important inference capability of a terminological representation system is *classification*. The classifier computes all *subsumption* relationships between concepts, i.e., the subconcept-superconcept relationships induced by the concept definitions. In this paper we consider optimizations for the classification process only. We do not take into account problems that are specific to assertional reasoning. This emphasis on the terminological component is partially justified by the fact that this is the part that participates in most reasoning activities of almost all systems—which means that the efficiency of this reasoning component is crucial for the overall behavior of the system.

The first terminological representation system, KL-ONE [7], was an implementation of Brachman's [5] work on structured inheritance networks. In the last decade many knowledge representation systems based on these ideas have been built, for example BACK [37], CLASSIC [36], KANDOR [34], KL-TWO [42], K-Rep [30], KRYPTON [6], KRIS [3], LOOM [29], MESON [13], NIKL [41], SB-ONE [23], and YAK [10]. Moreover, formal aspects of ter-

minological representation languages have been thoroughly investigated, with the highest emphasis having been placed on the decidability and complexity of the subsumption problem [24, 31, 39, 35, 33, 40, 11, 12]. As a result of these investigations, it is known that subsumption determination is at least NP-hard or even undecidable for most languages that appear to be useful in applications. The developers of terminological representation systems usually have reacted to this problem in one of the following two ways. On the one hand, there are systems such as CLASSIC which support only a very limited terminological language, but employ almost complete reasoning methods. On the other hand, systems such as LOOM provide for a very powerful language, but the reasoning is incomplete, which means that not all existing subsumption relationships are detected.

The only system that does not make this compromise, i.e., that provides complete algorithms for a very expressive concept description language, is KRIS. Obviously, this means that KRIS will need exponential time for worst case examples which cannot be represented in expressively limited systems or are not treated in a complete way by incomplete systems. However, it is not *a priori* clear whether this also implies that KRIS has to be less efficient for "typical" knowledge bases. In particular, it might at least be fast in cases where its full expressive power is not used, or where incomplete algorithms are still complete. The empirical analysis of terminological representation systems described by Heinsohn *et al.* [17, 18] seems to preclude this possibility, though. KRIS turned out to be much slower than, for example, CLASSIC, even for knowledge bases that use a subset of CLASSIC's concept language, and for which CLASSIC's subsumption algorithm is complete.

One aim of the present paper is to demonstrate that this bad performance of KRIS is not mainly due to the use of complete subsumption

algorithms, but instead to the fact that the tested version was the first implementation of an experimental system where efficiency considerations only played a minor role. For this purpose we shall consider possible optimizations of the classification process on three different levels. The optimizations on the *first level* are independent of the fact that what we are comparing are concepts defined by a terminological language. On this level, classification is considered as the abstract order-theoretic problem of computing a complete representation of a partial ordering (in our case the subsumption hierarchy) by making as few as possible explicit comparisons (in our case calls of the subsumption algorithm) between elements of the underlying set (in our case the set of all concepts occurring in the terminology). Optimizations on the *second level* still leave the subsumption algorithm unchanged, but they do employ the fact that we are not comparing abstract objects but instead structured concepts. At this level subsumption relationships that are obvious consequences of this structure can be derived without invoking the subsumption algorithm. On the *third level*, the actual subsumption algorithm is changed so that it can benefit from the information provided by subsumption relationships which have previously been computed. The effects these optimizations have on the classification process are evaluated on three different sets of test data, which are described below.

It should be noted that we do not claim that all the presented optimizations are novel. Similar optimizations can probably be found in many of the existing systems [27, 28, 38, 44]. Further, the optimizations on the first level described below are very similar to methods that can be found in the conceptual graphs literature [25, 14, 26], and which have been used in the implementation of the PEIRCE system [15].

However, until now it was not possible to find a coherent description of all the methods, and there were no empirical studies on their exact ef-

fects. A second motivation for this work is thus to make such a description available for implementors of terminological representation systems. For this reason, we describe the optimization techniques on an abstract, algorithmic level and evaluate their effects using implementation-independent metrics, i.e., instead of measuring runtime (in one particular system) we evaluate the complexity of the methods by counting specific operations that are supposedly very costly.

The paper is structured as follows. In the next section we introduce the basic notions of terminological representation and reasoning. Section 3 describes in some detail the test data we used to evaluate the different optimization methods. Sections 4 through 6 describe different methods to speed up classification on the three levels described above. Finally, in Section 7, we summarize our findings and describe the effects the optimizations had on the KRIS system.

2 Terminological Knowledge Representation and Reasoning

For the reader's convenience, this section provides a brief introduction into the area of terminological knowledge representation. Since we are only considering optimizations of the classification process, this introduction will be restricted to the terminological part of the system (for an introduction that includes the assertional part as well, see [4]). We define syntax and semantics of a prototypical terminological language, which includes most of the constructs occurring in our test knowledge bases. In addition, we introduce the reasoning service whose optimization is the topic of the present paper (namely, computing the subsumption hierarchy), and we sketch two types of subsumption algorithms used in terminological

systems.

The Terminological Language $\mathcal{ALCN}$

The terminological formalism can be used to describe knowledge about classes of individuals (concepts) and relationships between these classes.

Definition 2.1 (Syntax of $\mathcal{ALCN}$) Concept descriptions *are built from concept and role names using the concept-forming operators*

negation *($\neg C$)*, disjunction *($C \sqcup D$)*, conjunction *($C \sqcap D$)*,
existential restriction *($\exists R.C$)*, value restriction *($\forall R.C$)*,
at-least restriction *($\geq n.R$)*, at-most restriction *($\leq n.R$)*.

Here C and D are syntactic variables for concept descriptions, R stands for a role name, and n for a nonnegative integer. The set of concept names is assumed to contain the particular names $\top$ and $\bot$ for the top *and the* bottom *concept.*

Let A be a concept name and let D be a concept description. Then $A = D$ and $A \sqsubseteq D$ are terminological axioms. *A* terminology (TBox) *is a finite set $\mathcal{T}$ of terminological axioms with the additional restrictions that* (i) *each concept name (different from $\top$ and $\bot$) appears exactly once as a left hand side of an axiom, and* (ii) *$\mathcal{T}$ contains no cyclic definitions.*

At-least and at-most restrictions are also called number restrictions. An expression of the form $A = D$ is called a *complete definition* of A, and an expression of the form $A \sqsubseteq D$ is called a *primitive definition* of A. Depending on whether the name A is introduced by a complete or a primitive definition, A is called *defined* or *primitive concept*.

Intuitively, a cyclic definition is one that refers to itself. To make this more precise, we introduce the *definition order* of a TBox: We say that

a concept name A directly uses a concept name B in a TBox if B occurs in the description that defines A. Let "uses" be the transitive closure of "directly uses." Then A comes in the definition order after B if A uses B. The TBox does not contain cycles if this relation is irreflexive, i.e., if there is no concept name A such that A uses A.

As an example, consider the terminology of Figure 1. The concepts `Female` and `Human` are introduced without a restriction on their extension, whereas `Male` is restricted to be in the complement of `Female`. The concepts `Woman` and `Man` are not completely defined since being female (male) and human is not enough to define a woman (man). For example, a female toddler is usually not called woman. The concept names `Momo` and `Momm` are respectively acronyms for "Mother of males only" and "Mother of many males."

`preferred place for Figure 1`

Definition 2.2 (Semantics of $\mathcal{ALCN}$) *An* interpretation $\mathcal{I}$ *for* $\mathcal{ALCN}$ *consists of a set* $dom(\mathcal{I})$ *and an extension function that associates*

- *with each concept name* A *a subset* $A^{\mathcal{I}}$ *of* $dom(\mathcal{I})$,
- *with each role name* R *a binary relation* $R^{\mathcal{I}}$ *on* $dom(\mathcal{I})$, *i.e., a subset of* $dom(\mathcal{I}) \times dom(\mathcal{I})$.

The special names top and bottom are interpreted as $\top^{\mathcal{I}} = dom(\mathcal{I})$ *and* $\bot^{\mathcal{I}} = \emptyset$. *For* $x \in dom(\mathcal{I})$, *we denote the set* $\{y \in dom(\mathcal{I}) \mid xR^{\mathcal{I}}y\}$ *of* R*-fillers of* x *by* $R^{\mathcal{I}}(x)$.

The extension function can be extended to arbitrary concept descriptions as follows:

- $(\neg C)^{\mathcal{I}} = dom(\mathcal{I}) \backslash C^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, *and* $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$,
- $(\exists R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid R^{\mathcal{I}}(x) \cap C^{\mathcal{I}} \neq \emptyset\}$,

- $(\forall R.C)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\}$,
- $(\geq n.R)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid |R^{\mathcal{I}}(x)| \geq n\}$,
- $(\leq n.R)^{\mathcal{I}} = \{x \in dom(\mathcal{I}) \mid |R^{\mathcal{I}}(x)| \leq n\}$.

An interpretation $\mathcal{I}$ is a model *of the TBox $\mathcal{T}$ iff it satisfies $A^{\mathcal{I}} = D^{\mathcal{I}}$ for all terminological axioms $A = D$ in $\mathcal{T}$, and $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all terminological axioms $A \sqsubseteq D$ in $\mathcal{T}$.*

Using this model-theoretic semantics it is easy to define the inference services of terminological systems in a formal way. The terminological axioms of a given TBox imply subconcept-superconcept relationships (so-called subsumption relationships) between the concepts introduced in this terminology.

Subsumption: Let $\mathcal{T}$ be a TBox and let A, B be concept names occurring in $\mathcal{T}$. Then B *subsumes* A with respect to $\mathcal{T}$ (symbolically $A \sqsubseteq_{\mathcal{T}} B$) iff $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$.

When a terminological system reads in a terminology, it first computes all the subsumption relationships between the concepts introduced in this TBox. This process, called *classification*, results is an explicit internal representation of the subsumption ordering, which can directly be accessed during later computations, and which in most systems is displayed to the user by a graphical interface. Figure 2 gives a graphical representation of the subsumption ordering induced by the TBox of Figure 1.

`preferred place for Figure 2`

The subsumption algorithms described in the literature are usually not concerned with subsumption between concept names with respect to a TBox. They compute *subsumption relationships between concept*

descriptions, where the description D is said to subsume the description C iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. To use these algorithm for solving the subsumption problem with respect to a TBox, one must first eliminate primitive definitions that are not of the form $A \sqsubseteq \top$, and then expand complete definitions (see [32], Section 3.2.4 and 3.2.5).

In our example, the primitive definitions

$$\begin{array}{lcl} \texttt{Male} & \sqsubseteq & \neg\texttt{Female} \\ \texttt{Woman} & \sqsubseteq & \texttt{Human} \sqcap \texttt{Female} \\ \texttt{Man} & \sqsubseteq & \texttt{Human} \sqcap \texttt{Male} \end{array}$$

are replaced by

$$\begin{array}{lcl} \texttt{Male} & = & \neg\texttt{Female} \sqcap \texttt{Male*} \\ \texttt{Male*} & \sqsubseteq & \top \\ \texttt{Woman} & = & \texttt{Human} \sqcap \texttt{Female} \sqcap \texttt{Woman*} \\ \texttt{Woman*} & \sqsubseteq & \top \\ \texttt{Man} & = & \texttt{Human} \sqcap \texttt{Male} \sqcap \texttt{Man*} \\ \texttt{Man*} & \sqsubseteq & \top. \end{array}$$

Intuitively, the newly introduced concept name `Male*` (`Woman*`, `Man*`) stands for the absent part in the definition of `Male` (`Woman`, `Man`). After applying this replacement to each primitive definition with right-hand side different from $\top$, all remaining primitive definitions are of the form $A \sqsubseteq \top$. Since such an axiom does not restrict the extension of A in any way, primitive definitions need no longer be taken into account by the subsumption algorithms.

To get rid of complete definitions as well, they are expanded as follows: defined concepts are iteratively replaced by their defining descriptions until all names occurring on the right-hand side of a complete definition are

primitive. The expansion process terminates since TBoxes are assumed to be acyclic. In our example, the expanded definitions of `Man` and `Father` are

$$\begin{aligned} \texttt{Man} &= \texttt{Human} \sqcap \neg\texttt{Female} \sqcap \texttt{Male*} \sqcap \texttt{Man*} \\ \texttt{Father} &= \texttt{Human} \sqcap \neg\texttt{Female} \sqcap \texttt{Male*} \sqcap \texttt{Man*} \sqcap \exists\texttt{child.Human}. \end{aligned}$$

It is easy to see that `Father` is subsumed by `Man` with respect to the TBox of Figure 1 iff the subsumption relationship holds for the corresponding expanded descriptions, i.e., iff $\texttt{Human} \sqcap \neg\texttt{Female} \sqcap \texttt{Male*} \sqcap \texttt{Man*} \sqcap \exists\texttt{child.Human}$ is subsumed by $\texttt{Human} \sqcap \neg\texttt{Female} \sqcap \texttt{Male*} \sqcap \texttt{Man*}$.

Thus we have seen that subsumption with respect to a TBox can be reduced to subsumption of concept descriptions. It should, however, be noted that expanding concept definitions may cause an exponential growth of the knowledge base. This is the reason why subsumption with respect to a TBox is already intractable for very small sub-languages of $\mathcal{ALCN}$. In fact, Nebel [33] has shown that subsumption with respect to a TBox is co-NP-complete for the language that has conjunction and value restriction as its only concept constructors. In contrast, subsumption of concept descriptions of this language is still polynomial, as has been shown by Levesque and Brachman [24]. However, Levesque and Brachman also show that a relatively small extension of the language (namely, adding existential restrictions) makes subsumption of concept descriptions intractable as well.

The subsumption algorithms employed in terminological systems can be divided into two classes: structural algorithms and tableau-based algorithms. In the remainder of this section, we briefly explain the ideas underlying both types of algorithms.

Structural Subsumption Algorithms

These algorithms usually proceed in two phases. First, the descriptions to be tested for subsumption are normalized, and then the syntactic structure of the normal forms is compared.

For simplicity, we restrict our attention to descriptions containing conjunction and value restriction only. Such a description is in normal form iff it is of the form

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \ldots \sqcap \forall R_n.C_n,$$

where $A_1, \ldots, A_m$ are distinct concept names, $R_1, \ldots, R_n$ are distinct role names, and $C_1, \ldots, C_n$ are concept descriptions in normal form. It is easy to see that any description can be transformed into an equivalent[1] one in normal form, using associativity, commutativity and idempotence of $\sqcap$, and the fact that the descriptions $\forall R.(C \sqcap D)$ and $(\forall R.C) \sqcap (\forall R.D)$ are equivalent.

Now let

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \ldots \sqcap \forall R_n.C_n,$$

be the normal form of the description C, and let

$$B_1 \sqcap \ldots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_l.D_l,$$

be the normal form of the description D. Then C is subsumed by D iff the following two conditions hold:

1. For all $i, 1 \leq i \leq k$, there exists $j, 1 \leq j \leq m$ such that $B_i = A_j$.

2. For all $i, 1 \leq i \leq l$, there exists $j, 1 \leq j \leq n$ such that $S_i = R_j$ and

[1] The concept descriptions C and D are equivalent iff for all interpretations $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$.

C_j is subsumed by D_i.

It is easy to see that this subsumption algorithm is sound (i.e., the "if" direction of the above statement holds) and complete (i.e., the "only-if" direction of the above statement holds as well), and that its time complexity is polynomial (see [24] for a proof). This subsumption algorithm can relatively easily be extended to handle descriptions that contain number restrictions in addition to conjunction and value restrictions (see [32], Section 4.1.1). For larger languages, however, structural subsumption algorithms usually fail to be complete.

Tableau-based Subsumption Algorithms

Using a different paradigm, sound and *complete* subsumption algorithms have been developed for a large class of terminological languages (see, e.g., [40, 19, 12]). Usually, these algorithms are *satisfiability checking algorithms* rather than algorithms directly computing subsumption. They are model generation procedures, and are similar to first-order tableaux calculus, with the main difference that the specific structure of concept descriptions allows one to impose an appropriate control that ensures termination. Unlike structural algorithms, such tableau-based algorithms are also complete for languages that allow for negation, disjunction, and existential restrictions.

Since a concept description D subsumes a description C if, and only if, $C \sqcap \neg D$ is not satisfiable, i.e., there does not exist an interpretation that interprets $C \sqcap \neg D$ as a non-empty set, a satisfiability algorithm can indeed be used to solve the subsumption problem. In order to check whether a given concept description C is satisfiable, the tableau-based algorithm tries to generate a finite interpretation in which C is interpreted as a non-empty set. This generation process is complete in the sense that, if

it fails, i.e., an obvious contradiction occurs, we can conclude that C is not satisfiable; otherwise C is satisfiable.

In the following, we illustrate this type of algorithm by an example (a formal description of a tableau-based algorithm for $\mathcal{ALCN}$ can be found in [19]).

Let A, B be concept names, and let R be a role name. Assume that we want to know whether $(\exists R.A) \sqcap (\exists R.B)$ is subsumed by $\exists R.(A \sqcap B)$. That means that we have to check whether the concept description

$$C := (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$$

is not satisfiable.

In a first step, we push all negation signs as far as possible into the descriptions, using de Morgan's rules and the usual rules for quantifiers. As a result, we obtain the description

$$C' := (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B),$$

where negation occurs only in front of concept names.

In a second step we try to construct a finite interpretation $\mathcal{I}$ such that $C'^{\mathcal{I}} \neq \emptyset$. This means that there has to exist an individual in $dom(\mathcal{I})$ that is an element of $C'^{\mathcal{I}}$. The algorithm just generates such an individual, say b, and imposes the constraint $b \in C'^{\mathcal{I}}$ on it. Since C' is the conjunction of three concept descriptions, this means that b has to satisfy the following three constraints: $b \in (\exists R.A)^{\mathcal{I}}$, $b \in (\exists R.B)^{\mathcal{I}}$, and $b \in (\forall R.(\neg A \sqcup \neg B))^{\mathcal{I}}$.

From $b \in (\exists R.A)^{\mathcal{I}}$ we can deduce that there has to exist an individual c such that $(b, c) \in R^{\mathcal{I}}$ and $c \in A^{\mathcal{I}}$. Analogously, $b \in (\exists R.B)^{\mathcal{I}}$ implies the existence of an individual d with $(b, d) \in R^{\mathcal{I}}$ and $d \in B^{\mathcal{I}}$. We should not assume that $c = d$ since this would possibly impose too many constraints

on the individuals newly introduced to satisfy the existential restrictions on b. Thus the algorithm introduces for any existential restriction a new individual as role-filler, and this individual has to satisfy the constraints expressed by the restriction.

Since b must satisfy the value restriction $\forall R.(\neg A \sqcup \neg B)$ as well, and c, d were introduced as R-fillers of b, we also obtain the constraints $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ and $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$. Thus the algorithm uses value restrictions in interaction with already defined role relationships to impose new constraints on individuals.

Now $c \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ means that $c \in (\neg A)^{\mathcal{I}}$ or $c \in (\neg B)^{\mathcal{I}}$, and we have to choose one of these possibilities. If we assume $c \in (\neg A)^{\mathcal{I}}$ this clashes with the other constraint $c \in A^{\mathcal{I}}$, which means that this search path leads to an obvious contradiction. Thus we have to choose $c \in (\neg B)^{\mathcal{I}}$. Analogously, we have to choose $d \in (\neg A)^{\mathcal{I}}$ in order to satisfy the constraint $d \in (\neg A \sqcup \neg B)^{\mathcal{I}}$ without creating a contradiction to $d \in B^{\mathcal{I}}$. Thus, for disjunctive constraints, the algorithm tries both possibilities in successive attempts. It has to backtrack if it reaches an obvious contradiction, i.e., if the same individual has to satisfy constraints that are obviously conflicting.

In the example, we have now satisfied all the constraints without encountering an obvious contradiction. This shows that C' is satisfiable, and thus $(\exists R.A) \sqcap (\exists R.B)$ is not subsumed by $\exists R.(A \sqcap B)$.

The algorithm has generated an interpretation $\mathcal{I}$ as witness for this fact: $dom(\mathcal{I}) := \{b, c, d\}$; $R^{\mathcal{I}} := \{(b, c), (b, d)\}$; $A^{\mathcal{I}} := \{c\}$ and $B^{\mathcal{I}} := \{d\}$. For this interpretation, $b \in C'^{\mathcal{I}}$. This means that $b \in ((\exists R.A) \sqcap (\exists R.B))^{\mathcal{I}}$, but $b \notin (\exists R.(A \sqcap B))^{\mathcal{I}}$.

Termination of the algorithm is ensured by the fact that the newly introduced constraints are always smaller than the constraints which enforced their introduction.

3 The Test Data

In order to evaluate the different optimization techniques empirically, we used three sets of test data. As Heinsohn *et al.* [17, 18], we consider both existing knowledge bases used in other projects (six different KBs with the number of concepts ranging between 140 and 440), and randomly generated knowledge bases whose structure resembles those of the six real knowledge bases. Additionally, we also used randomly generated partial orders to evaluate different methods of computing the subsumption hierarchy.

Below we give a brief description of the six realistic knowledge bases. Table 1 characterizes the structure of the original KBs by means of the number of defined and primitive concepts and roles, respectively. As mentioned by Heinsohn *et al.* [17, 18], in the process of automatically translating and adapting the KBs to each particular system, some artificial concepts have been introduced, the cardinality is also shown in Table 1. The exact number of auxiliary concepts differ from system to system, though. The numbers given here are for the CLASSIC system. A more structural characterization of the subsumption hierarchy induced by the KBs is given in Table 2 in Section 4.4.

CKB (Conceptual Knowledge Base): Contains knowledge about tax regulations and is used in the Natural Language project XTRA at the University of Saarbrücken.

Companies: Contains knowledge about company structures and is used at the Technical University Berlin in the framework of the ESPRIT project ADKMS.

FSS (Functional Semantic Structures): Contains knowledge about speech acts and is used in the Natural Language project XTRA at

the University of Saarbrücken.

Espresso: Contains knowledge about Espresso machines and their structure. It is used in the WIP-Project of DFKI in the framework of multimodal presentation of information.

Wisber: Contains knowledge about different forms of investments and was used in the natural language dialog project WISBER at the University of Hamburg.

Wines simple kosher: Contains knowledge about wines, wineries, and meal-courses. It is used as sample KB of the CLASSIC system.[2]

`preferred place for Table 1`

The experiments by Heinsohn *et al.* [17, 18] have shown that besides the size of a knowledge base its structure can have a significant influence on the runtime necessary to classify the knowledge base. In order to determine how the effects of the optimizations vary with the size of the knowledge base, it is necessary to have access to a set of knowledge bases with a similar structure and varying size. The randomly generated knowledge bases used in the empirical evaluation of terminological representation systems [17, 18] have this property.[3] The structure of these randomly knowledge bases resembles the structure of realistic knowledge bases in many important aspects (see below) and, as has been shown empirically, leads to a runtime behavior of implemented terminological representation systems comparable to their performance on realistic knowledge bases [17, 18].

[2] A lot of individuals have been transformed to general concepts because in our tests we only considered terminological knowledge but did not want to cut all the nice information about different wineries and wines.

[3] For related work on random structures in the database area see [2, 9, 8].

The generated knowledge bases have the following properties:

- 80% of the concepts are primitive.
- There are exactly 10 different roles.
- Each concept definition is a conjunction containing
  - one or two concept symbols (explicit super-concepts),
  - zero or one at-least restrictions,
  - zero or one at-most restrictions,
  - and zero, one, or two value restrictions,

  where the number of constructs from one category and the roles and concepts are randomly assigned with a uniform distribution. Further, the concepts are constructed in a way such that no concept is unsatisfiable (i.e., no at-least restriction is larger than any at-most restriction).

In order to avoid definitional cycles, the concepts are partitioned into *layers*, where the ith layer has 3^i concepts. When assigning explicit super-concepts or value-restriction concepts to the concept definition of a concept from level i, only concepts from level 0 to $i-1$ are considered.

Since the first level of optimizations can be done in an abstract order-theoretic setting, these optimizations are also evaluated on randomly generated partial orderings. These partial orderings were generated as follows [43]. In order to generate a partial order $(\{1,\ldots,n\}, <_P)$:

1. Choose a positive integer k.
2. Generate randomly k permutations $\pi_i = (p_{1,i},\ldots,p_{n,i})$. Such a permutation defines a linear ordering $<_i$ on $\{1,\ldots,n\}$ as follows: $r <_i s$ iff r comes before s in π_i.

3. The strict partial ordering relation $<_P$ on $\{1, \ldots, n\}$ is now defined as: $r <_P s$ iff $r <_i s$ for all $i, 1 \leq i \leq k$.

Note that for $k = 1$, the resulting partial order is a total order. Further, for k approaching n, the generated partial orders tend to become flat, i.e., most elements will be pairwise incomparable. In our experiments, we varied the parameter k in order to evaluate the effect of the optimizations on the first level for varying structural properties of the ordering.

4 Avoiding Subsumption Calls when Computing the Subsumption Hierarchy

In the first level of optimizations we are concerned with computing the concept hierarchy induced by the subsumption relation. More abstractly, this task can be viewed as computing the representation of a partial ordering. For a given partial ordering[4] $\leq$ on some set P, $\prec$ shall denote the *precedence relation* of $\leq$, i.e., $\prec$ is the smallest relation such that its reflexive, transitive closure is identical with $\leq$. Obviously, $x \prec y$ iff $x \leq y$ and there is no z different from x and y such that $x \leq z \leq y$. If $x \leq y$, we say that x is a *successor* of y and y is a *predecessor* of x. Similarly, if $x \prec y$, we say that x is an *immediate successor* of y and y is an *immediate predecessor* of x.

Given a set X and a partial ordering $\leq$ on X, computing the representation of this ordering on X amounts to identifying $\prec$ on X. If $\leq$ is a *total* ordering, this task is usually called sorting. For a partial ordering it is called the identification problem [16]. The basic assumption here is that the partial ordering is given via a comparison procedure, and that the

[4] A partial ordering is a transitive, reflexive, and antisymmetric relation.

comparison operation is rather expensive. For this reason, the complexity of different methods to compute the precedence relation is measured by counting the number of comparisons.

In our case, X is the set of concepts defined in a terminological knowledge base, and $\leq$ is the subsumption relation between these concepts.[5] The assumption that the subsumption test is the most expensive operation is justified by the known complexity results for the subsumption problem [11]. It is also supported by the empirical fact that terminological representation systems (such as the optimized version of KRIS) spend more than 95% of their runtime on subsumption checking, even when dealing with typical knowledge bases that do not contain worst cases.

The worst case complexity of computing the representation of a partial ordering on a set with n elements is obviously $O(n^2)$ because it takes $n \times (n-1)$ comparisons to verify that a set of n incomparable elements is indeed a flat partial order. Since subsumption hierarchies typically do not have such a "pathological" structure, considerably less than $n \times (n-1)$ comparisons will almost always suffice.

Below, we describe and analyze four different methods to identify the representation of a partial ordering, namely, the *brute force* method, the *simple traversal* method, the *enhanced traversal* method, and the *chain inserting* method. Average case analyses of these methods seem to be out of reach since one does not know enough about the structure of "typical" terminological knowledge bases, and since it is not even known how many different partial orders exist for a given number of elements [1]. For this reason, the different methods are compared empirically.

All methods we describe are incremental, i.e., assuming that we have

[5] To be more precise, the subsumption relation is only a quasi-ordering, i.e., it need not be antisymmetric. For the following discussion, this is mostly irrelevant, however. There is only one place in the algorithms where this fact has to be taken into account.

identified the precedence relation $\prec_i$ for $X_i \subseteq X$, the methods compute for some element $c \in X - X_i$ the precedence relation $\prec_{i+1}$ on $X_{i+1} = X_i \cup \{c\}$. The two most important parts of this task are the *top search* and the *bottom search*. The top search identifies the *set of immediate predecessors* in X_i for a given element c, i.e., the set $X_i{\downarrow}c := \{x \in X_i \mid c \prec x\}$. Symmetrically, the bottom search identifies the *set of immediate successors* of c, denoted by $X_i{\uparrow}c$.

To be more precise, the procedures for top search that we will describe below compute the set $\{x \in X_i \mid c \leq x \text{ and } c \not\leq y \text{ for all } y \prec_i x\}$, which in most cases is the set $X_i{\downarrow}c$. Because the subsumption relation is only a quasi-ordering, there is one exception. The concept c can be equivalent to an element x of X_i, i.e., $c \leq x$ and $x \leq c$. In this case, the top search procedures will yield $\{x\}$ instead of $X_i{\downarrow}c$. To take care of this case, we test $x \leq c$ whenever the top search procedure yields a singleton set $\{x\}$. If this test is positive, c is equivalent to x, and we know that $X_i{\downarrow}c = X_i{\downarrow}x$, and $X_i{\uparrow}c = X_i{\uparrow}x$, which means that we don't need the bottom search phase. Otherwise, the result of the top search procedure is in fact $X_i{\downarrow}c$.

Given the set $X_i{\downarrow}c$, $X_i{\uparrow}c$, and $\prec_i$, it is possible to compute the precedence relation $\prec_{i+1}$ on $X_{i+1} = X_i \cup \{c\}$ in linear time. In fact, one just has to add $\prec$-links between c and each element of $X_i{\downarrow}c$, and between each element of $X_i{\uparrow}c$ and c. In addition, all $\prec$-links between elements of $X_i{\uparrow}c$ and $X_i{\downarrow}c$ have to be erased.

4.1 The Brute Force Method

The top search part of the brute force method can be described as follows:

1. Test $c \leq x$ for all $x \in X_i$.
2. $X_i{\downarrow}c$ is the set of all $x \in X_i$ such that the test succeeded and for all $y \prec_i x$ the test failed.

The bottom search is done in the dual way.

This method obviously uses $2 \times |X_i|$ comparisons for the step of inserting c in X_i. Summing over all steps leads to $n \times (n-1)$ comparison operations to compute the representation of a partial ordering for n elements. Further, this is not only the worst-case, but also the best-case complexity of this method.

4.2 The Simple Traversal Method

It is obvious that many of the comparison operations in the brute force method can be avoided. Instead of testing the new element c blindly with all elements in X_i, in the top search phase the partial ordering can be traversed top-down and in the bottom search phase bottom-up, stopping when immediate predecessors or successors have been found. This leads us to the specification of the simple traversal method.

The top search starts at the top[6] of the already computed hierarchy. For each concept $x \in X_i$ under consideration it determines whether x has an immediate successor y satisfying $c \leq y$. If there are such successors, they are considered as well. Otherwise, x is added to the result list of the top search.

In order to avoid multiple visits of elements of X_i and multiple comparisons of the same element with c, the top search algorithm described in Figure 3 employs one label to indicate whether a concept has been "visited" or not and another label to indicate whether the subsumption test was "positive," "negative," or has not yet been made. The procedure *top-search* gets two concepts as input: the concept c, which has to be inserted, and an element x of X_i, which is currently under consideration.

[6] We assume that our concept hierarchies always contain a top element $\top$ and a bottom element $\bot$.

For this concept x we already know that $c \leq x$, and *top-search* looks at its direct successors with respect to $\prec_i$. Initially, the procedure is called with $x = \top$. For each direct successor y of x we have to check whether it subsumes c. This is done in the procedure *simple-top-subs?*. Since our hierarchy need not be a tree, y may already have been checked before, in which case we have memorized the result of the test, and thus need not invoke the expensive subsumption procedure *subs?*. The direct successors for which the test was positive are collected in a list *Pos-Succ*. If this list remains empty, x is added to the result list; otherwise *top-search* is called for each positive successor, but only if this concept has not been visited before along another path.

`preferred place for Figure 3`

The bottom search can be done again in the dual way. The number of subsumption tests of the simple traversal method relative to the brute force method are displayed in Figure 4. In the case of realistic knowledge bases, 40–60% of all subsumption calls are avoided and the savings are even higher for the random knowledge bases. For instance, the top search phase of the simple traversal method needs only 1/5 of the subsumption calls that are required by the brute force method.

`preferred place for Figure 4`

It is interesting to note that this top search is in principle the same as the one described by Lipkis [27], who implemented the first classification algorithm for KL-ONE. The bottom search described by Lipkis, however, is more efficient than the one given here (but less efficient than the method we consider next).

4.3 The Enhanced Traversal Method

Although the simple traversal method is a big advantage compared with the brute force method (see Figure 4), it still does not exploit all possible information. First, during the top search phase, we can take advantage of tests that have already been performed. Second, in the bottom search phase, we can use the information gained during the top search as well.

Of course, a dual strategy is also possible, i.e., performing the bottom search before the top search and exploiting the information gathered during the bottom search phase. Analyzing Figure 4, it becomes quickly obvious that this strategy would be less efficient, however. In fact, for the simple traversal method—where the top and bottom phase are done in a symmetric way—the top search phase turns out to be a lot faster. Thus it is better to start with this phase because the information gained thereby can then be used to speed up the slower bottom search phase.

When trying to take advantage of tests that have already been performed during top search one can either concentrate on negative information (i.e., a subsumption test did not succeed) or on positive information (i.e., a subsumption test was successful).

To *use negative information* during the top search phase one has to check whether for some predecessor z of y the test $c \leq z$ has failed. In this case, we can conclude that $c \not\leq y$ without performing the expensive subsumption test [28]. In order to gain maximum advantage, all direct predecessors of y should have been tested before the test is performed on y [25]. This can be achieved by using a modified breadth-first search where the already computed hierarchy is traversed in topological order, as described by Ellis [14] and Levinson [26]. Alternatively, one can make a recursive call whenever there is a direct predecessor that has not yet been tested. This is what the procedure *enhanced-top-subs?* described

in Figure 5 does. If y is not yet marked, the procedure *enhanced-top-subs?* is recursively called for all direct predecessors z of y. As soon as one of these calls returns *false*, one goes to the "else" branch, and marks y "negative." Only if all calls return *true*, the subsumption test *subs?*(y,c) is performed to decide whether y has to be marked "positive" or "negative." If we replace the call of *simple-top-subs?* in *top-search* by a call of *enhanced-top-subs?*, we get the top search part of the enhanced traversal method.

`preferred place for Figure 5`

The enhanced top search procedure just described makes maximum use of failed tests. Alternatively, it is possible to *use positive information.* Before checking $c \leq y$, one can look for successors z of y that have passed the test $c \leq z$ [28]. If there exists such a successor, one can conclude that $c \leq y$ without performing an actual subsumption test. Although we are only interested in minimizing the number of comparison operations, it should be noted that instead of searching for a successor that has passed the test it is more efficient to propagate positive information up through the subsumption hierarchy. This can be achieved by an easy modification of the procedure *simple-top-subs?*. When the call *subs?*(y,c) yields *true*, not only y is marked "positive," but so are all of y's predecessors. Obviously, this technique cannot be combined with the enhanced top search described in Figure 5 since it reduces the number of subsumption tests only if there are predecessors which have not yet been tested, and enhanced top search tests all predecessors before making a subsumption test.

Both methods are obviously more efficient than simple traversal since it is guaranteed that they never make more subsumption tests than the simple traversal method. However, although both methods have been

mentioned in the literature, they have never been compared theoretically or empirically.

First, it can easily be shown that neither of these alternatives is uniformly better than the other one. This can be seen by considering the examples described in Figure 6 and 7.

`preferred place for Figure 6`

In the first example, the top-search using negative information makes $n+1$ tests: it first tests x_1, then goes to y, but before testing it, it tests all its direct predecessors, i.e., $x_2, \ldots, x_n$. The top search using positive information makes two tests: first x_1 and then y; the positive result of this second test is propagated to $x_2, \ldots, x_n$.

`preferred place for Figure 7`

In the second example, the top search using negative information needs only two tests: first it tests y_1, then goes to x_1, but before testing x_1 its direct predecessor y_2 is tested. The negative result of this test prevents $x_1, \ldots, x_n$ from being tested. The top search using positive information tests $n+2$ nodes: first y_1, then all its successors $x_1, \ldots, x_n$, and finally y_2.

However, we have observed significant performance differences for the two different top search strategies. For the random knowledge bases, the method using positive information was only slightly better than the simple traversal method (less than 5%). For this reason, we have also considered a "hybrid method" which propagates positive information up, and negative information down the hierarchy (but does not test all predecessors before testing a node). Propagating negative information down is again achieved by an easy modification of *simple-top-subs?*. When the call of *subs?*(y,c) yields *false*, not only y is marked "negative," but all

of y's successors. The hybrid method turned out to be a lot better than just propagating positive information, but it still needed slightly more tests (approx. 5%–10%) than the enhanced top search for all but one of the random knowledge bases. On five of the six realistic knowledge bases the hybrid method was insignificantly faster than the enhanced top search (less than 1%). On the remaining realistic KB, the hybrid method needed 10% more comparisons. Although these results do not seem to be conclusive in favor of the hybrid method or the enhanced top search, it is obvious that the use of negative information leads to a significantly greater reduction of comparisons than the use of positive information. These findings for terminological knowledge bases coincide with what has been observed by Levinson [26] for conceptual graphs.

Now we turn to the *bottom search* phase of the enhanced traversal method. Of course, optimizations dual to the ones described for the top search can be employed here. In addition, the set $X_i{\downarrow}c$ can be used to severely cut down the number of comparisons in the bottom search phase. As mentioned by Lipkis [27], the search for immediate successors of c can be restricted to the set of successors of $X_i{\downarrow}c$. In fact, the set of candidates for $X_i{\uparrow}c$ is even more constrained. Only elements that are successors of *all* $x \in X_i{\downarrow}c$ can be immediate successors of c [25, 14, 26]. This optimization is achieved by an easy modification of the procedure *enhanced-bottom-search* (which is dual to *enhanced-top-search*): the test "marked?(y,"negative")" is augmented to "marked?(y,"negative") or y is not a successor of all $x \in X_i{\downarrow}c$." The remaining problem is how to implement the second part of this test. One possibility is to mark the successors of the elements of $X_i{\downarrow}c$ in an appropriate way, and then test these labels [26]. Another possibility, which we have used in our tests, is to equip each concept in X_i with a list of all its predecessors in X_i, and test whether $X_i{\downarrow}c$ is contained in the list of predecessors of y.

As a result of this optimization, the number of necessary comparison operations can be cut down to a fraction compared with the simple bottom search strategy. Interestingly, we observed a further reduction of comparison operations in case of the real knowledge bases when searching top-down starting at $X_i{\downarrow}c$ instead of searching bottom-up. For the random knowledge bases, no such difference was observed, however. The bottom search described by Ellis [14] and Levinson [26] is also done top-down.

`preferred place for Figure 8`

The effects of the enhanced traversal method for the realistic and random knowledge bases as test data are displayed in Figure 8. Comparing these graphs with the graphs in Figure 4, the advantage of the enhanced traversal method over the simple traversal method becomes obvious. An interesting further phenomenon is that the relative savings of the enhanced traversal method increase with the number of concepts. It should be noted, however, that the improvement for the realistic KBs in the top search phase is not overwhelming. In fact, only 1–13% of the subsumption calls with respect to the simple traversal method are avoided, with a tendency of higher savings for larger knowledge bases.

Of course, the enhanced traversal method only pays off if the assumptions spelled out in the beginning of this section are not violated, i.e., if the subsumption costs are dominating the classification costs and are considerably higher than the costs incurred by the extra operations. Since the efficiency gains for the realistic KBs in the top search phase of the enhanced traversal method are not very large compared to the simple traversal method, there might be the question whether this optimization is really worthwhile.

Assuming that the average runtime costs of checking subsumption

between two concepts (including the traversal of the hierarchy) is r, and the number of concepts that are checked in the top search phase of the simple traversal method is n_s, then the overall costs for the top search of the simple traversal method are

$$t_s = n_s \times r.$$

Assuming that the average overhead of checking whether the predecessor have been tested is o for one concept and that n_e concepts are checked for subsumption with the new concept in the top search phase of the enhanced traversal method, the overall costs of the top search of the enhanced traversal method are

$$t_e = n_e \times r + n_s \times o.$$

Measuring the runtime of the subsumption test and the runtime for the test whether the predecessors have been already checked in KRIS on realistic and random knowledge bases reveals that an average subsumption test is 200 times slower than the latter test, i.e, $o = 0.005 \times r$, and thus

$$t_e = n_e \times r + n_s \times 0.005 \times r.$$

Hence, the top search of the enhanced traversal pays off if $n_e/n_s \leq 0.995$. For our test data, this relationship was always satisfied.

Since the overhead costs o are directly proportional to the number of direct predecessors, the overhead increases with the number of direct predecessors. The empirically justified assumption that $o = 0.005 \times r$ is only applicable to knowledge bases similar to those considered in our experiments. It may not be valid for knowledge bases with a higher average number of direct predecessors.

4.4 The Chain Inserting Method

The methods we have considered so far are based on finding the right spot where to insert the new concept by traversing the subsumption hierarchy top-down and bottom-up. Recalling that there is a tight connection between our problem of computing the representation of a partial ordering and the sorting problem, it may be worthwhile to have a closer look at sorting techniques.

Sorting a set of elements that is linearly ordered can be done either by incrementally searching the already ordered sequence linearly or by using binary search. In the former case, we inevitably end up with quadratic complexity, while in the latter case $O(n \times \log n)$ is a possibility. Of course, it seems attractive to transfer the latter technique to our problem, an idea that leads to the chain inserting method. This method is similar to Algorithm A described by Faigle and Turán [16]. However, the assumptions by Faigle and Turán [16] are somewhat different from ours. There it is assumed that a single test yields the answer "greater," "smaller," or "incomparable," whereas we would need two calls of the subsumption procedure to get this information.

In order to specify the chain inserting method, we first define the notion of a chain covering of a partial ordering. A *chain covering* is a partition of a partial ordering into *chains*, i.e., totally ordered subsets. Provided we have a chain covering of the set X_i, it is possible to identify the sets $X_i{\downarrow}c$ and $X_i{\uparrow}c$ by binary search in all chains. For a given chain C_j of the covering $X_i = C_1 \cup \cdots \cup C_m$, binary search is used to find the least predecessor and the greatest successor of c in C_j. Since the underlying ordering $\leq$ is only a partial ordering on X, the new element c to be inserted into the chain C_j need not be comparable with all elements of C_j. For this reason one needs two binary search phases for each chain.

The first one asks $c \leq x$, and treats negative answers as if they would mean $c > x$. This phase yields the least predecessor of c in C_j. The other phase is dual, and yields the greatest successor of c in C_j. The set of these least predecessor (resp. greatest successors) for all chains of the covering yields a superset of $X_i{\downarrow}c$ (resp. $X_i{\uparrow}c$). The set $X_i{\downarrow}c$ (resp. $X_i{\uparrow}c$) is obtained by eliminating the elements which are not minimal (resp. maximal) with respect to $\leq_i$. As a further optimization, propagation of positive and negative information of successful and of failed tests in the existing subsumption hierarchy is used to make some of the explicit subsumption tests during binary search superfluous, after one or more chains have already been searched through.

We have also considered a "hybrid" method that employs chain inserting for long chains and enhanced traversal afterwards. The idea here is that by binary search in long chains one gets rather quickly into the "center" of the partial ordering, from which propagation of positive and negative information should have the greatest effect.

It is, of course, advisable to use chain coverings with a minimal number of chains. Unfortunately, the computation of minimal chain-coverings is nontrivial and takes more than quadratic time [22]. Instead of using this quite expensive algorithm, we designed a simple heuristic method to compute almost minimal chain-coverings incrementally.

This heuristic proceeds as follows. When a new element c is inserted, the chain covering is updated as follows. After the sets $X_i{\downarrow}c$ and $X_i{\uparrow}c$ have been computed, c is inserted in the longest chain satisfying one of the following conditions:

1. Binary search has yielded both a least predecessor and greatest successor in the chain, and they are successive elements of the chain. In this case, c is inserted between these two elements in the chain.

2. Binary search has yielded a least predecessor (or greatest successor) in the chain, and it is the least (resp. greatest) element of the chain. In this case, c is inserted below (resp. above) this element in the chain.

If there is no chain satisfying one of these conditions, a new chain consisting of c is created.

In our experiments, the chain coverings obtained this way were close to minimal. They always contained at most 10% more chains than the minimal chain-coverings.

We expected that the chain-inserting method would outperform the enhanced traversal method. However, to our surprise, the chain inserting method turned out to be not significantly better than the enhanced traversal method. To the contrary, on the realistic KBs it is usually less efficient, except for one case, and the same holds for the random KBs. The "hybrid" version using chain inserting for long chains and enhanced traversal afterwards was also not much better than the pure chain-inserting method.

An explanation for this poor behavior of the chain-inserting method could be that in typical knowledge bases most of the chains in a chain covering are very short and that the effect of propagating positive and negative information is very limited because the *connectivity* of the knowledge bases, i.e., the average number of reachable predecessors and successors, is small.

In order to test this conjecture, it is necessary to run the algorithm on knowledge bases with different *structural properties* than those present in the realistic and random knowledge bases. For this purpose, we used random partial orders generated according to the method described in Section 3. These additional experiments showed that the chain-inserting

method is indeed sometimes more efficient than the enhanced traversal method.

The empirical results concerning the performance of the chain inserting method relative to the enhanced traversal method are given in Table 2. We have only displayed the results for top search, since the bottom search is almost identical if the optimizations from the enhanced traversal are included. The first group of results was obtained by applying the chain-inserting method to the realistic KBs, the second group gives the results for the random KBs, and the third group specifies the result for the randomly generated partial orders.

`preferred place for Table 2`

In addition to the size of the partial order (first column) and the relative number of comparison operations with respect to the enhanced traversal method (last column), also some structural parameters of the partial orders are given. The second column gives the average number of immediate predecessors and successors (where the top and bottom elements are not counted). The third column gives the average number of successors and predecessors, and the fourth and fifth column specify the breadth and depth (including top and bottom), respectively, of the partial order, where *breadth* corresponds to the cardinality of the chain covering and *depth* corresponds to the longest chain in the chain covering.

As already mentioned, the chain-inserting method does more subsumption tests on the realistic and random knowledge bases, but is sometimes more efficient on random partial orders. In fact, the results of our experiments on random partial orders are consistent with the conjecture that high connectivity and long chains will lead to a relative performance gain over the enhanced traversal method.

The chain-inserting method may thus become more interesting for

knowledge bases defining relatively deep hierarchies with high connectivity. Additionally, the chain-inserting method may prove to be worthwhile if the transitive closure of the precedence relation $\prec$ is implemented using storage compression techniques based on chain coverings as described by Jagadish [21]. Finally, it should be noted that the overhead of the chain-inserting method is not significantly higher than the overhead of the enhanced traversal method. In fact, in our implementation the chain-inserting method required slightly less overhead than the enhanced traversal method.

5 Exploiting Obvious Subsumption Relationships

In this section we describe some further techniques for avoiding subsumption tests by exploiting relations which are obvious when looking at the syntactic structure of concept definitions.[7] These pre-tests require only little effort but can speed up the classification process significantly. We consider three different optimizations, which can be used at different stages of the classification process. All three techniques apply only if the descriptions of the concepts are conjunctive (which is the case for the majority of concepts, in particular if we consider the existing real knowledge bases).

The first technique can be used prior to the top search. Assume that we are inserting the (conjunctive) concept c into the already computed part of the hierarchy. If the description defining c mentions x explicitly as a conjunct, then it is obviously the case that $c \leq x$. We call such concepts x *told subsumers* of c. Of course, if x is also a conjunctively defined

[7] These techniques are probably used in all systems, see, e.g. [38].

concept, it may have told subsumers as well, and these (and their told subsumers, etc.) can be included into the list of told subsumers of c. It is rather easy to compile this list while reading in the concept definitions. The information that c is subsumed by its told subsumers can be propagated through the existing hierarchy $(X_i, \prec_i)$ prior to the top search, e.g., by pre-setting the markers used in the traversal method to "positive" for the told subsumers and all their predecessors. A prerequisite for this optimization technique to be effective is that the told subsumers of c are already contained in X_i. This can be achieved by inserting concepts following the definition-order (see Section 2 for the definition of this order).

The second optimization technique is applicable if concepts are conjunctive and are inserted in the subsumption hierarchy following the definition-order. In this case, the bottom search phase can completely be avoided if a *primitive concept* (i.e., a concept that is introduced by a primitive definition, which gives only necessary conditions) has to be classified. In fact, such a concept c can only subsume the bottom concept and concepts for which c is a told subsumer. Since the second type of possible subsumees consists of concepts whose definitions use c, and which are thus not yet present in the actual hierarchy when inserting along the definition-order, the result of the bottom search is just the bottom concept $\bot$. Considering the fact that in realistic KBs the majority of concepts (60%-90%) are primitive, this optimization can save most of the subsumption calls during the bottom search phase. Combining the two optimization techniques led to a saving of 10% to 20% with respect to the pure enhanced traversal method for the realistic knowledge bases. In case of the random knowledge bases, the savings where even greater, as can be seen from Figure 9.

preferred place for Figure 9

A final optimization technique can be used as a pre-test before calling the subsumption algorithm. As mentioned in Section 2, a given TBox containing primitive definitions can be transformed into one where all primitive definitions have $\top$ as right-hand side. For a conjunctive concept c, the *primitive components* of c are the told subsumers that are introduced by such a primitive definition. By extracting and caching the primitive components of all concepts, it becomes possible to check whether a subsumption relation is possible by comparing the sets of primitive components: c can only be subsumed by d if the set of primitive components of d is a subset of the set of primitive components of c. Thus, if the subset test gives a negative result, the subsumption algorithm need not be called. Although such a test overlaps with computations the subsumption algorithm does, it is much faster than the subsumption test. For this reason, this pre-test pays off if most of the subsumption calls can be avoided, which was indeed the case for our test data. Our experiments indicate that the number of calls of the subsumption algorithm can be again reduced by 50%-60%, if this technique is applied.

6 Speeding up the Subsumption Test

In this section we consider two possible optimizations of the subsumption algorithm, and describe the effects they have on the performance of classification for our test knowledge bases. As mentioned in Section 2 there are two different types of subsumption algorithms employed in terminological systems: *structural algorithms*, which are implemented in almost all terminological representation systems (e.g. CLASSIC, LOOM, BACK), and *tableau-based algorithms* as realized in KRIS. For conceptual simplic-

ity both types of subsumption algorithms are usually described in the literature as taking concept descriptions as arguments.[8] In this setting, the exploitation of previously computed subsumption relationships between concepts defined in the terminology is precluded since these concept names no longer occur in the descriptions.

6.1 The Optimizations

However, almost all terminological representation systems take advantage of previously computed subsumption relationships, i.e., they make use of subsumption relationships that have already been computed and stored during the classification process.

To illustrate how this can be done for the structural subsumption algorithm described in Section 2, suppose that C and D are descriptions in normal form, i.e., C (D) is a conjunction of distinct concept names A_i (B_j) and value restrictions $\forall R_k.C_k$ ($\forall S_l.D_l$) with distinct role names, where each C_k (D_l) is again in normal form. The test whether C is subsumed by D recursively calls a subsumption test for the descriptions C_k and D_l (if the roles R_k and S_l are identical). In case C_k and D_l are concept names of possibly defined concepts, and we already know whether there is a subsumption relationship between C_k and D_l, the recursive call of the subsumption algorithm can be replaced by a simple table look-up. Thus, it is rather natural and straightforward to incorporate the use of already computed subsumption relations into a structural subsumption algorithm. It should be noted that it is an essential requirement *not* to expand the concept definitions before checking subsumption since otherwise the concept names for which subsumption relationships are already

[8] Recall that subsumption with respect to acyclic TBoxes can be reduced to subsumption of such concept descriptions by expanding the concept definitions (cf. Section 2).

known would be lost. Further, it is necessary to classify the concepts according to the "definition-order" mentioned in the previous section.

In contrast to other terminological systems, KRIS employs a satisfiability algorithm to determine subsumption relationships between concepts. Since a satisfiability algorithm does not recursively call subsumption algorithms but satisfiability algorithms, it is not obvious how to exploit previously computed subsumption relationships. A closer look, however, reveals that a satisfiability algorithm may detect a contradiction earlier during model generation if previously computed subsumption relationships are taken into account. To see this, suppose that we already know that a defined concept A subsumes a defined concept B. If during the model generation an element is constrained to be both instance of $\neg A$ and B, a contradiction can be detected without expanding the definitions of A and B. Again, this approach only works if the concept definitions are not expanded before starting to check satisfiability.

If expansion is done "by need" during the satisfiability test, one has to decide in which order to expand the concept names. It is easy to see that this order may have considerable impact on the runtime behavior. For example, assume that we are testing $A \sqcap B$ for satisfiability where in the TBox A is defined by a very large concept description and B is defined to be $\neg A \sqcap C$. If B is expanded first, the contradiction between A and $\neg A$ is detected at once. On the other hand, if A is expanded first, detecting the contradiction between the large descriptions associated with A and its negation may be rather time-consuming, depending on the structure of the description.

One way of avoiding this problem is to expand concept names according to the inverse of their definition-order, which in the above example would mean that we expand B before A, because the definition of B refers to A. Of course, this means that for each expansion operation one has to

go through the list of all expandable names, and look for a maximal one with respect to the definition-order. For our tests we have used another solution, which avoids searching for a maximal name, but may use more space. Here one expands in arbitrary order, but when a name is expanded it is not removed, but just marked as expanded. If, in our example, A is expanded before B, we then still have the name A, and as soon as B is expanded it yields the contradiction with $\neg A$.

In order to gain experience in how to optimize the satisfiability algorithm to be employed in KRIS, we implemented the following three versions.

1. The first one takes *expanded* concept descriptions as input. Since these descriptions do not contain names of defined concepts, obvious contradictions can only be detected between primitive concepts.

2. The second one *successively expands* the concept descriptions during model generation, but keeps the names, as described above. This allows the algorithm to detect obvious contradictions not only between primitive concepts but also between names of defined concepts.

3. The third version is a refinement of the second one in that already computed subsumption relationships are taken into account when looking for obvious contradictions.

6.2 Empirical Results and Analysis

It turns out that the first version is significantly slower than the second one, a result we did expect. The main reason for this behavior is that the number of recursive calls of the satisfiability algorithm is reduced due to obvious contradictions detected between names of defined concepts. As

a consequence, the runtime of the second version is reduced by 40-60% relative to the first version (see Figure 10, which displays the results for the random knowledge bases).

`preferred place for Figure 10`

A result we did *not* expect is that the behavior of the third version is no better than of the second, which means that trying to exploit already computed subsumption relationships does not pay off. The reason for this behavior seems to be that—at least for the test data—only a few contradictions are detected by using already computed subsumption relationships. This is indicated by the fact that the number of recursive calls of the satisfiability algorithm does not significantly decrease when going from the second to the third version. However, the test of whether a set of negated and unnegated concept names is contradictory w.r.t. already computed subsumption relationships is more complex than just searching for complementary names, which explains that the third version's runtime behavior is even slightly worse than the second one's (see Figure 10).

This result is all the more surprising since using computed subsumption relationships during classification is an optimization technique employed by most terminological systems. The reason why it may pay off for other systems could be that these systems first normalize, and during this normalization phase some of the systems introduce auxiliary concepts. For example, assume that C is defined by the description $\forall R.A \sqcap \forall R.B$, and D by $\forall R.A$. The normalization procedure as described in Section 2 replaces $\forall R.A \sqcap \forall R.B$ by $\forall R.(A \sqcap B)$. Instead, one can introduce a new concept name E, define it as $A \sqcap B$, and modify the definition of C to $\forall R.E$. Now the subsumption relationship between A and the auxiliary concept E—which is found first if the terminology is classified according to the definition-order—immediately entails that D subsumes C. Thus

classification of the terminology with the auxiliary concepts allows one to exploit previously computed subsumption relationships more often. On the other hand, it has the disadvantage that in general a lot more concepts have to be classified.

Another interesting behavior we observed is due to the interaction between different optimization techniques. The optimizations described in the previous two sections try to avoid subsumption tests, whereas the present section is concerned with speeding up the subsumption test. Ideally, one could expect that these optimizations are independent. This means that the overall speedup factor is the product of the speedup factors of the individual optimizations. This can only be true if the optimizations apply uniformly to all situations, however.

If the optimizations apply to special cases only, subsumption avoidance optimizations and subsumption test optimization may aim at similar special cases and lead to the situation that subsumption tests are avoided which have neglectable computational costs in any case.

If we take the second or third version's satisfiability algorithm, the exploitation of obvious subsumption relationships caused by *conjunctive definitions,* i.e., the first optimization technique mentioned in Section 5, does no longer speed up the classification process significantly. This is due to the fact that such subsumption relationships can now be easily detected by the satisfiability algorithms. For example, let C be a concept that is defined to be the conjunction of $C_1, \ldots, C_m$, where the C_i are defined concepts as well. The obvious subsumption relationship between C_i and C is immediately detected by the second and third version of the satisfiability algorithm, due to an obvious contradiction between C_i and $\neg C_i$.

7 Conclusion

We have described and analyzed different optimization techniques for the classification process in terminological representation systems. Interestingly, two of the most promising techniques, namely, the chain inserting method for computing the representation of a partial order and the exploitation of already computed subsumption relations in the subsumption algorithm, did not lead to the expected performance increase in case of realistic knowledge bases.

As a result of our empirical analysis, the optimization techniques that came off best were incorporated in the KRIS system. Whereas the unoptimized version was orders of magnitude slower than the fastest system tested by Heinsohn *et al.* [17, 18], the new version has now a runtime behavior similar to that of the other systems on the test data used there.

`preferred place for Figure 11`

Figure 11(a) displays the runtime of the new KRIS version for the realistic knowledge bases and contrasts them with the runtime figures given by Heinsohn *et al.* [17, 18]. Figure 11(b) gives the results for large random knowledge bases.[9]

It should be noted, however, that all the knowledge bases used in the test are formulated using quite limited terminological languages. An interesting open problem is the development of further optimization techniques for more powerful terminological languages containing also disjunction and negation and of specific optimization techniques for assertional reasoning.

[9]The description of the runtime behavior of the systems by Heinsohn *et al.* [17, 18] refers to system versions as of 1990 and does not necessarily reflect the performance of more recent versions.

Acknowledgements

We would like to thank Uwe Utsch for implementing the different subsumption strategies, Hans-Jürgen Bürckert, Jochen Heinsohn, Armin Laux, and Werner Nutt for helpful discussions concerning the topics described in this paper, and Alex Borgida, Peter Patel-Schneider, and the anonymous referees for helpful comments on an earlier version of this paper.

This work has been supported by the German Ministry for Research and Technology (BMFT) under research contracts ITW 8901 8 and ITW 8903 0 and by the Italian National Research Council (CNR), project "Sistemi Informatici e Calcolo Parallelo."

References

[1] M. Aigner. *Combinatorical Search.* Teubner, Stuttgart, Germany, 1988.

[2] T. L. Anderson, A. J. Berre, M. Mallison, H. H. Porter, III, and B. Schneider. The hypermodel benchmark. In *Proceedings of Extended Database Technology.* Springer-Verlag, 1990.

[3] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, June 1991.

[4] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In M. Richter and H. Boley, editors, *International Workshop on Processing Declarative Knowledge*, volume 567 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, 1991.

[5] R. J. Brachman. *A Structural Paradigm for Representing Knowledge.* PhD thesis, Havard University, 1977.

[6] R. J. Brachman, V. Pigman Gilbert, and H. J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON. In IJCAI-85 [20], pages 532–539.

[7] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, Apr. 1985.

[8] M. J. Carey, D. J. Dewitt, and J. F. Naughton. The OO7 benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Mangement of Data*, Washington, D.C., May 1993.

[9] R. G. G. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, 17(1):1–31, Mar. 1992.

[10] R. Cattoni and E. Franconi. Walking through the semantics of frame-based description languages: A case study. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent systems*, Knoxville, TN, Oct. 1990. North-Holland.

[11] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pages 151–162, Cambridge, MA, Apr. 1991. Morgan Kaufmann.

[12] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 458–465, Sydney, Australia, Aug. 1991. Morgan Kaufmann.

[13] J. Edelmann and B. Owsnicki. Data models in knowledge representation systems: A case study. In C.-R. Rollinger and W. Horn, edi-

tors, *GWAI-86 und 2. Österreichische Artificial-Intelligence-Tagung*, pages 69–74, Ottenstein, Austria, Sept. 1986. Springer-Verlag.

[14] G. Ellis. Compiled hierarchical retrieval. In *6th Annual Conceptual Graphs Workshop*, 1991.

[15] G. Ellis and R. A. Levinson. The birth of PEIRCE: A conceptual graphs workbench. In *Proceedings of the Seventh Annual Conceptual Graphs Workshop*, Las Cruces, New Mexico, July 1992.

[16] U. Faigle and G. Turàn. Sorting and recognition problems for ordered sets. *SIAM Journal of Computing*, 17(1):100–113, 1988.

[17] J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An empirical analysis of terminological representation systems. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence*, pages 767–773, San Jose, CA, July 1992. MIT Press.

[18] J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 1993. To appear. A preliminary version is available as DFKI Research Report RR-92-16.

[19] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 348–353, Stockholm, Sweden, Aug. 1990. Pitman.

[20] *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, CA, Aug. 1985.

[21] H. V. Jagadish. A compressed transitive closure technique for efficient fixed-point query processing. In L. Kerschberg, editor, *Expert Database Systems—Proceedings From the 2nd International Conference*, pages 423–446, Menlo Park, CA, 1989. Benjamin/Cummings.

[22] D. Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI Wissenschaftsverlag, Mannheim, Germany, 2nd edition, 1990.

[23] A. Kobsa. First experiences with the SB-ONE knowledge representation workbench in natural-language applications. *SIGART Bulletin*, 2(3):70–76, June 1991.

[24] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[25] R. A. Levinson. A self-organizing pattern retrieval system for graphs. In *Proceedings of the 4th National Conference of the American Association for Artificial Intelligence*, pages 203–206, Austin, TX, 1984.

[26] R. A. Levinson. Pattern associativity and the retrieval of semantic networks. *Journal of Computers & Mathematics with Applications*, 23(6–9):573–600, 1992.

[27] T. Lipkis. A KL-ONE classifier. In J. G. Schmolze and R. J. Brachman, editors, *Proceedings of the 1981 KL-ONE Workshop*, pages 128–145, Cambridge, MA, 1982. The proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

[28] R. MacGregor. A deductive pattern matcher. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence*, pages 403–408, Saint Paul, MI, Aug. 1988.

[29] R. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, June 1991.

[30] E. Mays, R. Dionne, and R. Weida. K-Rep system overview. *SIGART Bulletin*, 2(3):93–97, June 1991.

[31] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, Apr. 1988.

[32] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[33] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[34] P. F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 11–16, Denver, Colo., 1984. An extended version including a KANDOR system description is available as AI Technical Report No. 37, Palo Alto, CA, Schlumberger Palo Alto Research, October 1984.

[35] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272, June 1989.

[36] P. F. Patel-Schneider, D. L. McGuinness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3):108–113, June 1991.

[37] C. Peltason. The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119, June 1991.

[38] C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK system revisited. KIT Report 75, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, Sept. 1989.

[39] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference*, pages 421–431, Toronto, ON, May 1989. Morgan Kaufmann.

[40] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[41] J. G. Schmolze and W. S. Mark. The NIKL experience. *Computational Intelligence*, 6:48–69, 1991.

[42] M. B. Vilain. The restricted language architecture of a hybrid representation system. In IJCAI-85 [20], pages 547–551.

[43] P. Winkler. Random orders. *Order*, 1:317–331, 1985.

[44] W. A. Woods. Understanding subsumption and taxonomy: A framework for progress. In J. F. Sowa, editor, *Principles of Semantic Networks*, pages 45–94. Morgan Kaufmann, San Mateo, CA, 1991.

$$
\begin{array}{lcl}
\texttt{Human} & \sqsubseteq & \top \\
\texttt{Female} & \sqsubseteq & \top \\
\texttt{Male} & \sqsubseteq & \neg\texttt{Female} \\
\texttt{Woman} & \sqsubseteq & \texttt{Human} \sqcap \texttt{Female} \\
\texttt{Man} & \sqsubseteq & \texttt{Human} \sqcap \texttt{Male} \\
\texttt{Mother} & = & \texttt{Woman} \sqcap \exists \texttt{child}.\texttt{Human} \\
\texttt{Father} & = & \texttt{Man} \sqcap \exists \texttt{child}.\texttt{Human} \\
\texttt{Parent} & = & \texttt{Father} \sqcup \texttt{Mother} \\
\texttt{Grandmother} & = & \texttt{Woman} \sqcap \exists \texttt{child}.\texttt{Parent} \\
\texttt{Momo} & = & \texttt{Mother} \sqcap \forall \texttt{child}.\texttt{Male} \\
\texttt{Momm} & = & \texttt{Momo} \sqcap \geq 5.\texttt{child}
\end{array}
$$

Figure 1: A family terminology.

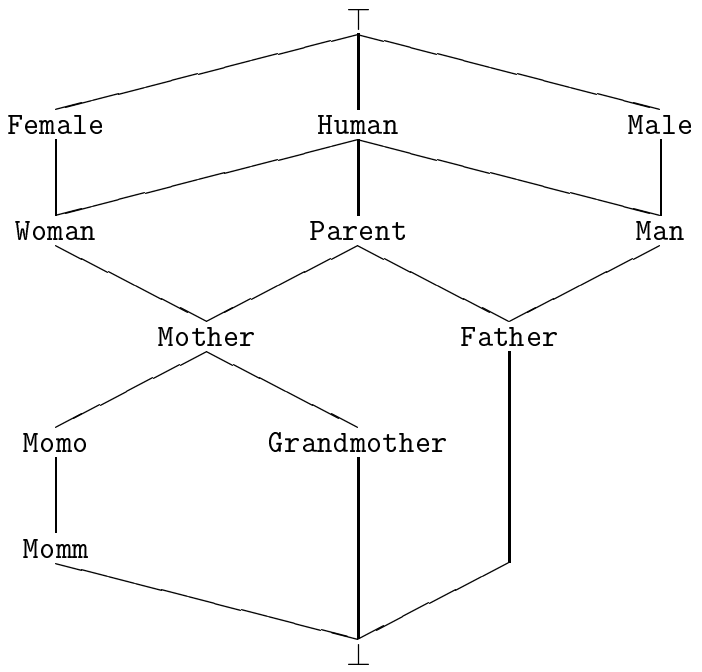


Figure 2: Subsumption hierarchy induced by the TBox of Figure 1.

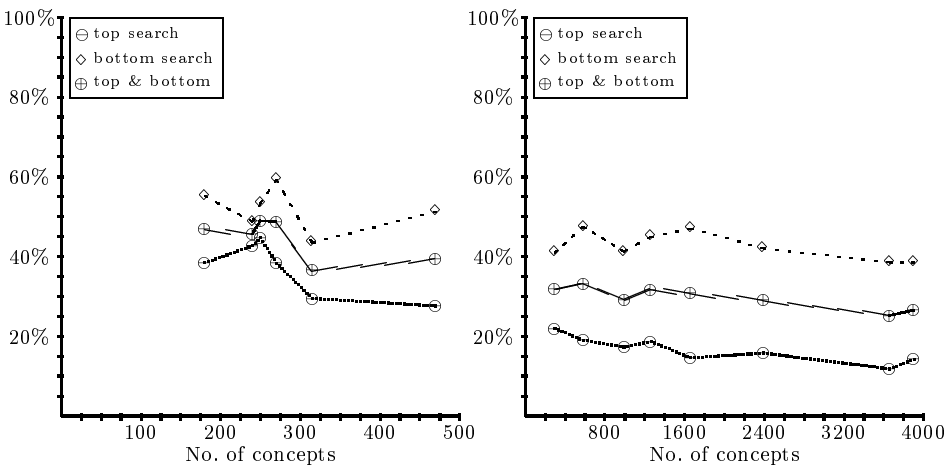
Name	defined	primitive	artificial	$\sum$	defined	primitive
	concepts				roles	
CKB	23	57	104	184	2	46
Companies	70	45	126	241	1	39
FSS	34	98	122	254	0	47
Espresso	0	145	124	269	11	41
Wisber	50	81	199	330	6	18
Wines	50	148	282	480	0	10

Table 1: Real Knowledge Bases: Structural description

```
top-search(c,x) =
    mark(x,"visited")
    Pos-Succ ← ∅
    for all y with y ≺_i x do
        if simple-top-subs?(y,c)
          then Pos-Succ ← Pos-Succ ∪ {y}
        fi
    od
    if Pos-Succ is empty
      then return {x}
      else  Result ← ∅
            for all y ∈ Pos-Succ do
                if not marked?(y,"visited")
                  then Result ← Result ∪ top-search(c,y)
                fi
            od
            return Result
    fi


simple-top-subs?(y,c) =
   if marked?(y,"positive")
     then return true
     elsif marked?(y,"negative")
           then return false
           elsif  subs?(y,c)
                  then mark(y,"positive")
                       return true
                  else  mark(y,"negative")
                       return false
           fi
     fi
   fi
```

Figure 3: Top search phase of the simple traversal method



(a) Realistic KBs (b) Random KBs

Figure 4: Number of comparison operations of the *simple traversal* method relative to the *brute force* method (vertical axis) against number of concepts (horizontal axis)

```
enhanced-top-subs?(y,c) =
    if marked?(y,"positive")
      then return true
      elsif marked?(y,"negative")
            then return false
            elsif for all z with y ≺_i z
                        enhanced-top-subs?(z,c)
                  and   subs?(y,c)
                  then mark(y,"positive")
                        return true
                  else  mark(y,"negative")
                        return false
            fi
      fi
    fi
```

Figure 5: Top search phase of the enhanced traversal method. The procedure *top-search* is adopted from the simple traversal method, but instead of *simple-top-subs?* it calls *enhanced-top-subs?*

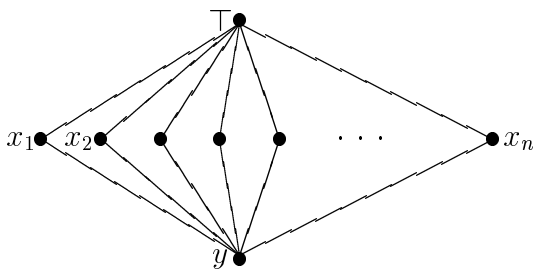


Figure 6: The new element c is a direct successor of y

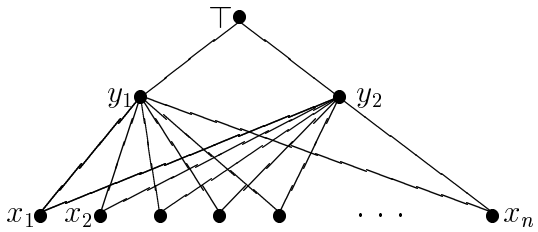
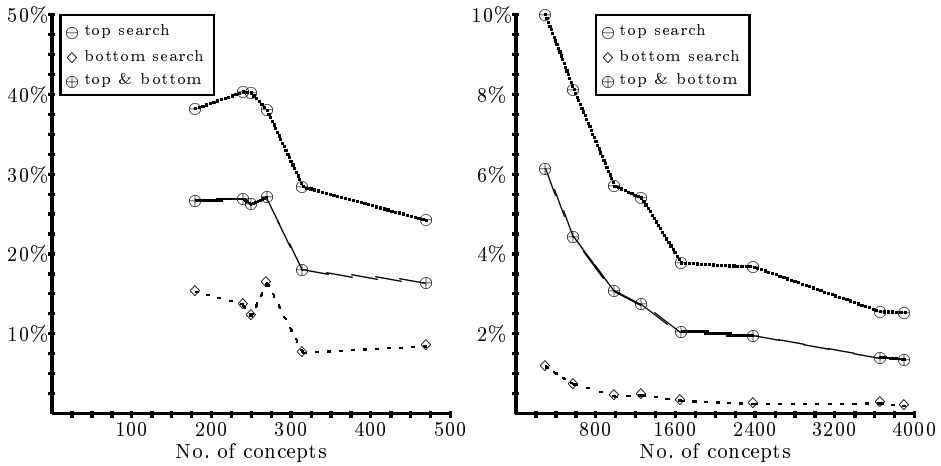


Figure 7: The new element c is a direct successor of y_1, but not a successor of y_2, $x_1, \ldots, x_n$



(a) Realistic KBs (b) Random KBs

Figure 8: Number of comparison operations of the *enhanced traversal* method relative to the *brute force* method (vertical axis) against number of concepts (horizontal axis)

No. of nodes	Average degree	Average no. of pred. & succ.	Breadth	Depth	Relative no. of comparisons
184	1.71	5.67	105	6	103.7%
241	1.91	6.38	124	6	100.3%
254	1.99	13.02	135	6	91.8%
269	1.72	5.16	164	7	107.9%
330	1.85	8.13	141	12	110.0%
298	2.36	8.88	142	8	115.7%
583	2.58	12.24	330	7	114.5%
992	2.73	16.77	478	10	111.7%
1263	3.18	16.61	661	11	108.9%
1659	3.19	18.86	927	10	110.3%
2389	3.50	25.49	1188	10	111.3%
3658	3.82	27.20	1703	8	105.3%
3905	4.04	33.95	1858	11	99.9%
301	7.67	42.11	88	8	73.2%
301	8.01	20.69	136	5	100.5%
301	6.40	10.43	168	6	102.7%
301	4.22	5.68	205	4	101.3%
586	9.93	72.55	144	9	67.2%
586	12.08	38.79	224	7	96.3%
586	10.39	20.42	301	7	103.3%
586	7.72	11.50	353	5	102.9%
995	5.52	250.24	85	28	16.2%
995	12.46	125.94	226	11	51.8%
995	16.40	62.88	354	9	91.4%
995	13.58	28.38	506	6	105.1%
1266	5.78	321.19	100	30	12.9%
1266	13.82	169.95	259	13	44.2%
1266	18.22	76.87	438	9	89.8%
1266	17.82	41.70	592	6	101.1%

Table 2: Number of comparison operations in top search of the chain-inserting method *relative* to enhanced traversal. The first group gives results for the realistic KBs, the second group for the random KBs, and the third group for the randomly generated partial orders.

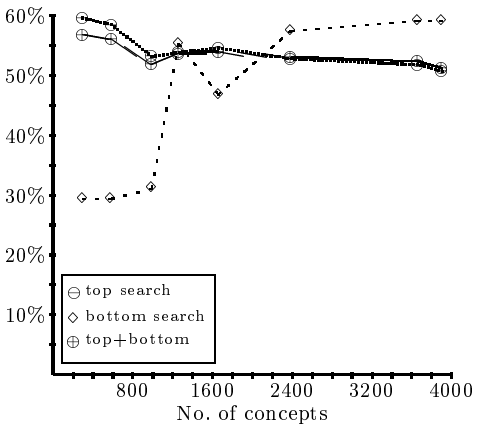


Figure 9: Number of necessary comparisons when exploiting obvious subsumption relations relative to pure *enhanced traversal* method for random KBs

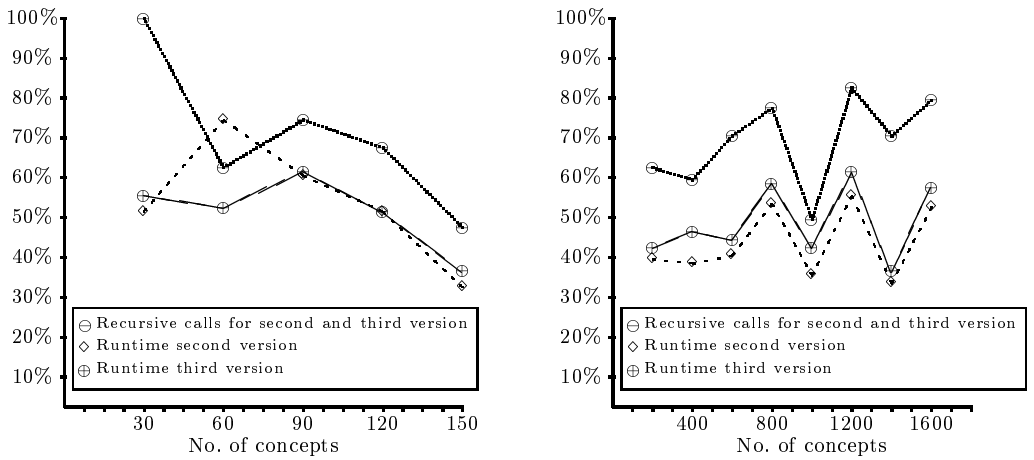
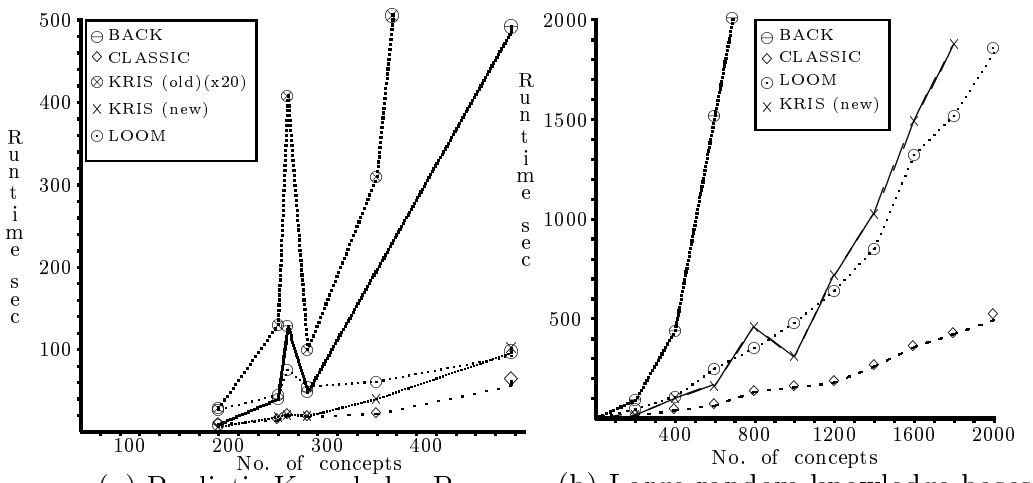


Figure 10: Runtime and number of recursive calls of the second and third version's satisfiability algorithm relative to the algorithm taking expanded concept terms as input (first version) for random KBs



(a) Realistic Knowledge Bases (b) Large random knowledge bases

Figure 11: Runtime performance for realistic and large random knowledge bases

List of Tables

1 Real Knowledge Bases: Structural description 48

2 Number of comparison operations in top search of the chain-inserting method *relative* to enhanced traversal. The first group gives results for the realistic KBs, the second group for the random KBs, and the third group for the randomly generated partial orders. 53

List of Figures

1 A family terminology. 47

2 Subsumption hierarchy induced by the TBox of Figure 1. . 47

3 Top search phase of the simple traversal method 49

4 Number of comparison operations of the *simple traversal* method relative to the *brute force* method (vertical axis) against number of concepts (horizontal axis) 50

5 Top search phase of the enhanced traversal method. The procedure *top-search* is adopted from the simple traversal method, but instead of *simple-top-subs?* it calls *enhanced-top-subs?* 51

6 The new element c is a direct successor of y 51

7 The new element c is a direct successor of y_1, but not a successor of y_2, $x_1, \ldots, x_n$ 52

8 Number of comparison operations of the *enhanced traversal* method relative to the *brute force* method (vertical axis) against number of concepts (horizontal axis) 52

9 Number of necessary comparisons when exploiting obvious subsumption relations relative to pure *enhanced traversal* method for random KBs 54
10 Runtime and number of recursive calls of the second and third version's satisfiability algorithm relative to the algorithm taking expanded concept terms as input (first version) for random KBs 54
11 Runtime performance for realistic and large random knowledge bases 55