Efficient Algorithms for θ-Subsumption

Tobias Scheffer and **Ralf Herbrich** and **Fritz Wysotzki**
Technische Universität Berlin, Artificial Intelligence Research Group, FR 5-8, Franklinstr. 28/29, D-10587 Berlin, email: scheffer@cs.tu-berlin.de

Abstract

θ subsumption is a decidable but incomplete approximation of logic implication, important to inductive logic programming and theorem proving. We show that by context based elimination of possible matches a certain superset of the determinate clauses can be tested for subsumption in polynomial time. We discuss the relation between subsumption and the clique problem, showing in particular that using additional prior knowledge about the substitution space only a small fraction of the search space can be identified as possibly containing globally consistent solutions, which leads to an effective pruning rule. We present empirical results, demonstrating that a combination of both of the above approaches provides an extreme reduction of computational effort.

1. Introduction

θ-subsumption (Robinson, 1965) is a correct but incomplete,*e.g.,* decidable consequence relation, while implication is undecidable in general. A clause C θ-subsumes D ($C \vdash_\theta D$), iff there is a substitution θ, such that $C\theta \subseteq D$ and $|C| \leq |D|$.

The two main fields of application of θ-subsumption are inductive logic programming (ILP) and theorem proving. The objective of ILP is to generate a hypothesis in a first-order language (*i.e.,* a logic program) that explains a set of positive examples and does not explain a set of negative samples. θ-subsumption is used as a consequence relation in many ILP systems, for the decision if a rule covers an example as well as for the reduction of clauses (*e.g.,* Muggleton and Feng (1990), van der Laag and Nienhuys-Cheng (1993), DeRaedt and Bruynooghe (1993)). In particular, the consistency test (*i.e.,* the test if a newly generalized clause covers negative samples) requires an extremely large amount of subsumption tests, which makes subsumption a bottleneck of ILP.

In theorem proving, θ-subsumption is used as a redundancy test in the connection graph proof procedure (Kowalski, 1975; Sickel, 1976), the elimination of clauses that are copies of already deduced clauses provides a strong reduction of the search space (Kim & Cho, 1992).

Decision of θ-subsumption of two clauses is NP-complete in general (Kapur & Narendran, 1986), even if the second clause is fixed (Kietz & Lübbe, 1994); the NP-completeness results from the ambiguity of variable identification. As subsumption is crucial to the power of ILP learners and theorem provers many approaches to speeding up subsumption have been studied.

If a clause includes a literal can be that matches exactly one literal of the other clause, this literal can be matched *deterministically* (Dzeroski et al., 1992; Kietz & Lübbe, 1994) and no back-tracking needs to be done. The complexity grows exponentially with the number of remaining, non-determinate literals only. In this paper, we will propose to reduce the number of candidates for each literal using contextual information.

For the similar problem of graph isomorphism, there are several approaches, (Tinhofer, 1976; Weisfeiler, 1976; Wysotzki et al., 1981; Unger & Wysotzki, 1981; Geibel & Wysotzki, 1996), to reducing matching candidates using context information. We will adapt a very general approach to the problem of subsumption, and show that characteristic matrices (Socher, 1988) are a special case of this approach. We will characterize the set of clauses that can be tested for subsumption in polynomial time by this algorithm.

Eisinger (1981) introduces *S-links* into the framework of the connection graph resolution proof procedure (Kowalski, 1975; Sickel, 1976). Eisinger further points out that a subsuming substitution exists, if there is a strongly compatible tuple of substitutions in the cartesian product of the literal matches. Kim and Cho (1992) propose a pruning strategy that reduces the computational effort of finding a compatible substitution. The maximum clique problem is strongly related to the subsumption problem. The clique problem is to find the largest subset of mutually adjacent nodes in a graph. This problem is well known to be NP-complete (*e.g.,* Feige et al. (1991)), however, much effort has been spent in the search for algorithms that behave efficient on the average (*e.g.,* Johnson and Trick (1996), Carraghan and Pardalos (1990), Gibbons et al. (1996)).

There is a different approach to reducing the complexity of subsumption, that can be combined with all previously mentioned approaches: If a clause contains classes *(locals)* of literals, such that there are no common variables in different classes, then each class can be matched independently and the complexity grows exponentially with the size of the largest local only (Gottlob & Leitsch, 1985; Kietz & Lübbe, 1994).

In Section 3, we extend the concept of deterministic matching, proposing to use contextual information in order to reduce the number of possible matches. In Section 4, we describe our clique based approach to subsumption, and in Section 5 we present our empirical results.

2. The θ-subsumption problem

θ-subsumption (Robinson, 1965; Plotkin, 1970) is an approximation of logical implication. A clause C θ-subsumes a clause D, written $C \vdash_\theta D$, iff there is a substitution θ, such that $C\theta \subseteq D$ and $|C| \leq |D|$. We use the term subsumption instead of θ-subsumption (in contrast to Loveland (1978), who defines subsumption as implication).

While implication is undecidable in general for first-order languages, θ-subsumption is decidable but incomplete – *i.e.,* there may exist clauses C and D, such that $C \not\vdash_\theta D$ but $C \models D$. This occurs when C is self-resolving (recursive) or if D is tautological (Gottlob, 1987). If tautologies and self-resolution are excluded, then $C \vdash_\theta D \Leftrightarrow C \models D$ (Gottlob, 1987; Muggleton, 1993; Kietz & Lübbe, 1994).

The θ-subsumption problem is NP-complete in general (Kapur & Narendran, 1986). Known algorithms have a worst case time complexity of $O(vars(D)^{vars(C)})$, or $O(|D|^{|C|})$.

Definition 1 (Substitution) *A substitution is a mapping from variables to terms. We denote substitutions $\theta = \{t_1/x_1, \ldots, t_n/x_n\}$, substituting a clause $C\theta$ rewrites the variables x_i by terms t_i.*

Definition 2 (Matching substitution) *A matching substitution from a literal l_1 to a literal l_2 is a substitution μ, such that $l_1\mu = l_2$.*

Definition 3 (Matching candidates) *The matching candidates of a literal $l_C \in C$ is the subset of a clause D, such that there is a matching substitution μ with $l_C\mu = l_D$ and $l_D \in D$.*

2.1 Deterministic subsumption

One approach to cope with the NP-completeness of θ-subsumption is the deterministic subsumption. A clause is said to be determinate, if there is exactly one possible match for each literal that is consistent with the previously matched literals (Muggleton & Feng, 1990), or, more generally, if there is an ordering of literals, such that in each step there is a literal which has exactly one match that is consistent with the previously matched literals (Kietz & Lübbe, 1994). However, determinacy is not an intrinsic property of a clause – a pair of clauses may or may not be testable for subsumption deterministically. In general, there may be only a few literals, or none at all, that can be matched deterministically. The remaining literals set up a – possibly smaller – search space.

Definition 4 (deterministic subsumption) *Let $C = c_0 \leftarrow c_1, \ldots, c_n$ and $D = d_0 \leftarrow d_1, \ldots, d_m$ be Horn clauses. C deterministically θ-subsumes D, written $C \vdash_{\theta DET} D$ by $\theta = \theta_0\theta_1 \ldots \theta_n$, iff there* *exists an ordering $c_{p(1)}, \ldots, c_{p(n)}$ of the c_i, such that for all i, $1 \leq i \leq n$ there exists exactly one θ_i with $\{c_{p(1)}, \ldots c_{p(i)}\}\theta_0 \ldots \theta_i \subseteq D$.*

$C \vdash_{\theta DET} D$ can be tested with at most $O(|C|^2|D|)$ unification attempts (Kietz & Lübbe, 1994) by the following algorithm:

1. While there is a literal $l_1 \in C$ that matches exactly one literal $l_2 \in D$ with $l_1\mu = l_2$, substitute C with μ.
2. If there is a literal in C that does not match any literal in D, then $C\theta \not\subseteq D$
3. If any literals could not be matched uniquely, start with the clause substituted so far and test for subsumption using a backtracking algorithm.

The main problem of this approach is that most pairs of clauses do not meet the determinacy condition. In our experiments we learned that almost no literal at all could be matched deterministically, unless the data set was especially prepared. However, the complexity is reduced dramatically for negative examples ($C \not\vdash_\theta D$). If the number of matching candidates for some literals in C can be reduced, as is done by the following algorithm, the condition holds more often and the set of clauses that can be tested for subsumption in polynomial time grows.

3. Context based candidate elimination

Wysotzki et al. (1981), Unger and Wysotzki (1981) propose an approach to reducing the number of matching candidates for the graph isomorphism problem that is based on results of Weisfeiler (1976) and Tinhofer (1976). It was applied to obtaining attribute-value data for machine learning of graph classification rules (Wysotzki et al., 1981; Geibel & Wysotzki, 1996). The approach is based on the idea that nodes may only be matched to those nodes that possess the same context – *i.e.,* the same relations up to an arbitrary depth. We will propose an approach to the subsumption problem that reflects this principle. Wysotzki's algorithm is based on the algebraic representation of graphs by adjacency matrices. Each element A_{ij} of an adjacency matrix contains the relation between node i and node j. The context of the nodes is computed by multiplying the adjacency matrices.

Example 1 *Let G_1 contain the nodes x_1, x_2 and x_3 labeled with the unary relation a and the relations $r(x_1, x_2)$ and $r(x_2, x_3)$. We can represent G_1 by the adjacency matrix*

$$A_1 = \begin{pmatrix} a & r & \emptyset \\ \emptyset & a & r \\ \emptyset & \emptyset & a \end{pmatrix}$$

The square of this matrix is:

$$A_1 \times A_1 = \begin{pmatrix} aa + r\emptyset + \emptyset\emptyset & ar + ra + \emptyset\emptyset & a\emptyset + rr + \emptyset a \\ \emptyset a + a\emptyset + r\emptyset & \emptyset r + aa + r\emptyset & \emptyset\emptyset + ar + ra \\ \emptyset a + \emptyset\emptyset + a\emptyset & \emptyset r + \emptyset a + a\emptyset & \emptyset\emptyset + \emptyset r + aa \end{pmatrix}$$

The elements of this matrix can be interpreted as follows: While element A_{ij} contains the relation between node i and node j, element A^2_{ij} of the multiplied matrix contains a complete enumeration of all paths of length 2 leading from node i to node j with each summand representing one path. In general, A^n_{ij} enumerates all paths of length n. We now take a look at element A^2_{11}, enumerating paths from node x_1 to node x_1: The first summand is aa, representing a path that consists of two loops through the unary relation a. The next summand is $r\emptyset$, representing the relation r leading to node x_2 and the empty relation $\emptyset$ back; this might not be considered a path in an intuitive sense because the empty relation was traversed, but it certainly starts from node x_1 and leads to node x_1. We now take a look at element A^2_{12}. Summand ar represents a path that is set up by a loop through the unary a and the binary $r(x_1, x_2)$ relation. Hence, the path leads from x_1 to x_2. The other summands are to be interpreted accordingly.

The branching factor of the search for an isomorphism can be reduced by only matching those nodes that share the same set of paths of length k for an arbitrary k. Note that the number of paths is $|V|^k$, where V is the set of nodes and k is the context depth. The comparison of paths is $O(paths^2) = O(|V|^{2k})$, but since k is an arbitrary (small) constant this is feasible and it turns out that almost any isomorphism test can be performed in polynomial time (Weisfeiler, 1976; Wysotzki et al., 1981; Unger & Wysotzki, 1981). It is known that two nodes cannot match if their context paths do not match, but it is unknown whether a match of the context implies a possible match of the nodes for any fixed context depth (note that the complexity of the graph isomorphism problem is unknown).

Translating this approach to the problem of θ-subsumption, we have to realize two main differences: The concept of a path does not cover more than binary relations and we want to decide whether one clause is a subset of another clause (up to substitution) rather than the identity of clauses (up to substitution). The translation is based on the observation that for each occurrence of a variable x_1 in C there must be a corresponding occurrence of the same $x_1\theta$ in D. We define the occurrence graph to denote the occurrence of identical variables. We firstly focus on datalog clauses, *i.e.,* clauses with functor-free terms.

Definition 5 (occurrence graph) (C, E_C) *is the occurrence graph of a datalog clause* C *with* $(l_i, l_j, \pi_i \rightarrow \pi_j) \in E_C$ *iff there is a variable* x *that occurs in literal* l_i *at argument position* π_i *and in literal* l_j *at argument position* π_j*.*

The edges of the occurrence graph are labeled $\pi_i \rightarrow \pi_j$, where π_i and π_j are argument positions in which occurrences of the same variable are found.

Definition 6 (context) *The context of depth* d *of a literal* l_1 *in a clause* C*,* $con(l_1, d, C)$*, is the set of terms* $p_1 \cdot \pi_1 \rightarrow \pi_2 \cdot p_2 \cdot \ldots \cdot \pi_{(n-1)} \rightarrow \pi_d \cdot p_d$*, iff there exists a set* $\{(l_1, l_2, \pi_1 \rightarrow \pi_2), \ldots, (l_{d-1}, l_d, \pi_{d-1} \rightarrow \pi_d)\}$ *of edges in the occurrence graph, and each* p_i *is the predicate symbol of* l_i *(note that* $l_i = l_j$ *is possible, too).*

Example 2 *Let* $C = r(x_1, x_2), r(x_2, x_3), q(x_3)$*. The occurrence graph contains four edges:* $(r(x_1, x_2), r(x_2, x_3), 2 \rightarrow 1)$*, indicating that there is a variable, that occurs on position 2 of literal* $r(x_1, x_2)$ *and on position 1 of literal* $r(x_2, x_3)$*, and* $(r(x_2, x_3), q(x_3), 2 \rightarrow 1)$*, corresponding to a variable on position 2 of* $r(x_2, x_3)$ *that also occurs on position 1 of* $q(x_3)$*, plus the symmetric correspondences* $(r(x_2, x_3), r(x_1, x_2), 1 \rightarrow 2)$ *and* $(q(x_3), r(x_2, x_3), 1 \rightarrow 2)$*. The context of literal* $r(x_1, x_2)$ *at depth 1 contains only the term* $r \cdot 2 \rightarrow 1 \cdot r$*. At depth 2 the context contains two paths:* $r \cdot 2 \rightarrow 1 \cdot r \cdot 2 \rightarrow 1 \cdot q$ *and* $r \cdot 2 \rightarrow 1 \cdot r \cdot 1 \rightarrow 2 \cdot r$*. The first path corresponds to the literal sequence* $r(x_1, x_2) - r(x_2, x_3) - q(x_3)$*, the second one to the "round trip" sequence* $r(x_1, x_2) - r(x_2, x_3) - r(x_1, x_2)$*.*

Proposition 1 *Let* $l_1 \in C$*,* $l_2 \in D$ *be literals, let the depth* d *be any natural number. Let* $l_1\mu = l_2$*,* μ *is a matching substitution. If* $con(l_1, d, C) \not\subseteq con(l_2, d, D)$*, then there is no* θ*, such that* $C\mu\theta \subseteq D$*.*

I.e., a literal cannot be matched against another literal within a globally consistent match, if its context cannot be embedded in the other literal's context.

Outline of proof 1 *Let* C *and* D *be clauses, let* $con(l_1, d, C)$ *and* $con(l_2, d, D)$ *be the context of a literal from* C *and* D *respectively, such that* $con(l_1, d, C) \not\subseteq con(l_2, d, D)$*. This implies, that there is a sequence of literals of* C *sharing at least one variable with their neighbors that has no correspondence in* D*. Let* $p_a \cdot \pi_a \rightarrow \pi_b \cdot p_b$ *be the critical part of the path, and* l_a*,* l_b *the pair of literals, that has* *no corresponding path in D. If there is a θ, such that $C\theta \subseteq D$, then $\pi_a(l_a)\theta = \pi_b(l_b)\theta$ where π is the argument selector, because the variables at these positions are equal. But $l_a\theta$ and $l_b\theta$ cannot be element of D, because they share a common variable and we assumed that there is no corresponding path in D.*

The graph context of a literal at depth 1 contains the same amount of information as the characteristic matrix (Socher, 1988) of the literal does. Element C_{ij} of a characteristic matrix of a literal l contains the predicate names of those literals l_k, such that there is a variable that occurs at position i of l and on position j of l_k. The graph context of depth 1 in turn contains a path for each pair of literals in which a common variable occurs, consisting of the predicate name and a term $i \rightarrow l_k$, such that the variable occurs at position i and j in either literal. Clearly, this incorporates the same information. The graph context at a larger depth contains more information, namely information about literals that are connected via a chain of common variables. This cannot be represented in the characteristic matrix formalism, because the matrix is indexed with two argument positions, whereas it requires a sequence of pairs of argument positions to represent context information of a higher depth.

The proposed algorithm reduces the number of literals in D which match a focused literal in C. There are two interesting cases in which the subsumption can be tested with polynomial effort: If there is a literal $l \in C$ that has no matching candidates left in D, then $C\theta \not\subseteq D$, and if there is a literal that has exactly one candidate, we need not backtrack the match in case of a failure, because there is no alternative. If we find a literal with only one remaining matching candidate in each step, $C \vdash_\theta D$ can be tested in $O(|C|^2|D|2^{2d})$, because $|C|^2|D|$ is the complexity of deterministic subsumption (Kietz & Lübbe, 1994) and we need to compare $O(2^d)$ paths to test for $con(l_i) \subseteq con(l_j)$, where d is the fixed depth.

Definition 7 (Context based determinacy) *Let $C = c_0 \leftarrow c_1, \ldots, c_n$ and $D = d_0 \leftarrow d_1, \ldots, d_m$ be Horn clauses and let k be the maximum number of literals in any literal's context of an arbitrary lookahead depth d. Then C $con(k,d)$-deterministically subsumes D by $\theta = \mu_0 \ldots \mu_n$, written $C \vdash_{\theta k d DET} D$, iff there exists an ordering $c_{p(1)}, \ldots, c_{p(n)}$, such that for all i, $1 \leq i \leq |C|$, there exists exactly one μ with $\{c_{p(1)}, \ldots c_{p(i)}\}\mu_0 \ldots \mu_i \subseteq D$ and $\forall j, 1 \leq j \leq i : con(c_{p(i)}, C, d) \subseteq con(c_{p(j)}\mu_j, D, d)$.*

Clearly, this is a generalization of the determinacy concept in the sense that if a clause C subsumes a clause D deterministically, it also subsumes D $con(k,d)$-deterministically. For the context depth of 0, $con(c_i, C, d)$ is the empty set and the definition of context based determinacy becomes equivalent to the definition of deterministic subsumption. For any context depth $d > 0$, and any context size $k > 0$, the context inclusion is an additional condition that reduces the number of candidates, and hence there exists more often at most one remaining matching candidate.

Example 3 *Let $C = r(x_1, x_2), r(x_2, x_3)$ and $D = r(y_1, y_2), r(y_2, y_3), r(y_1, y_3)$. We want to test if $C\theta \subseteq D$. Note, that the clauses cannot be matched deterministically. At depth 1, the context of the first literal $r(x_1, x_2)$ only contains the path $r \cdot 2 \rightarrow 1 \cdot r$ (x_2 appears at position 2 of $r(x_1, x_2)$ and on position 1 of $r(x_2, x_3)$), the context of $r(x_2, x_3)$ contains $r \cdot 1 \rightarrow 2 \cdot r$. The context of $r(y_1, y_2)$ is $\{r \cdot 2 \rightarrow 1 \cdot r, r \cdot 1 \rightarrow 1 \cdot r\}$, the context of $r(y_2, y_3)$ is $\{r \cdot 1 \rightarrow 2 \cdot r, r \cdot 2 \rightarrow 2 \cdot r\}$ and of $r(y_1, y_3)$ is $\{r \cdot 1 \rightarrow 1 \cdot r, r \cdot 2 \rightarrow 2 \cdot r\}$. Now the context of $r(x_1, x_2)$ can only be embedded in the context of $r(y_1, y_2)$, not in any other literal's context; the context of $r(x_2, x_3)$ is only included in the context of $r(y_1, y_2)$. Hence, both literals can be matched deterministically and the substitution was found without backtracking.*

If a clause C is not a datalog clause (*i.e.,* the clause contains terms with functor-symbols), we compute a datalog clause C' according to Socher (1988) and generate the context with respect to the new datalog clause. For each literal $p(t_1, \ldots, t_n) \in C$, C' contains a literal $p'(x_1, \ldots, x_m)$, such that the x_i are the variables occurring in the t_i, in order of their appearance.

4. Clique and the general subsumption problem

In the previous section, we discussed an algorithm that reduces the possible matching candidates of literals. In many cases, this procedure makes variables become deterministically matchable. In some cases, however, there is a remaining space of substitutions that has to be searched for a globally consistent substitution. In this section, we identify further restrictions of this remaining search space. We discuss the relation between subsumption and the clique problem, and show that a subsumption problem can be mapped to a clique problem with an equally large space. Additional prior knowledge about this space in conjunction with the pruning strategy of Carraghan and Pardalos (1990) will allow us exclude the largest part of it a priori. We will show that the remaining space is a proper subset of the search space that the Kim and Cho (1992) exhausts. We demonstrate empirically that actually only a small fraction of the space remains.

Definition 8 (Graph) *A pair (V, E) of vertices and edges with $E \subseteq V \times V$ is a graph.*

Definition 9 (Clique) *A graph $(C, C \times C)$ with $C \subseteq V$ is a* clique *of a graph (V, E), iff $E \supseteq C \times C$, i.e. all nodes are mutually adjacent.*

4.1 S-link method of subsumption

Eisinger (1981) proposes a subsumption test that is based on selecting a compatible tuple of substitutions.

Definition 10 (Compatibility) *Two substitutions θ_1 and θ_2 are called* strongly compatible *iff $\theta_1\theta_2 = \theta_2\theta_1$ (i.e., no variable is assigned different terms in θ_1 and θ_2).*

Definition 11 (Matching substitution) *$uni(C, l_i, D) = \{\mu | l_i \in C, l_i\mu \in D\}$ is the set of all matching substitutions from a literal l_i in C to some literal in D.*

Proposition 2 (Eisinger) *A clause C subsumes a clause D ($C\theta \subseteq D$), iff there is an n-tuple $(\theta_1, \ldots, \theta_n) \in \times_{i=1}^{n} uni(C, l_i, D)$, where $n = |C|$, such that all θ_i are pairwise strongly compatible.*

Example 4 *Let $C = \{P(x, y), P(y, z)\}$ and $D = \{P(a, b), P(b, c), Q(d)\}$. Then $uni(C, P(x, y), D) = \{\{a/x, b/y\}, \{b/x, c/y\}\}$ and $uni(C, P(y, z), D) = \{\{a/y, b/z\}, \{b/y, c/z\}\}$. The cartesian product of these sets is $\times_{i=1}^{2} uni(C, l_i, D) = \{\{a/x, b/y\}\{a/y, b/z\}, \{a/x, b/y\}\{b/y, c/z\}, \{b/x, c/y\}\{a/y, b/z\}, \{b/x, c/y\}\{b/y, c/z\}\}$, of which only $\{a/x, b/y\}\{b/y, c/z\} = \{a/x, b/y, c/z\}$ is a strongly compatible substitution.*

In order to test if there is a compatible substitution, the cartesian product of all matching substitutions has to be enumerated. There are $|D|^{|C|}$ combinations of matching substitutions in the worst case. We now map this problem to the clique problem. We therefore define a graph the nodes of which are all matching substitutions from any literal of C to some literal in D, and the edges of which are given by the compatibility of the substitutions. We augment the substitution with the number of the originating literal in C because we want each clique to contain only one matching substitution for each literal of C.

Definition 12 (substitution graph) *Let C and D be clauses and $n = |C|$. Then $(V_{C,D}, E_{C,D})$ is the substitution graph with $V_{C,D} = \bigcup_{i=1}^{n} (uni(C, l_i, D) \times \{i\})$ and $((\theta_1, i), (\theta_2, j)) \in E_{C,D}$ iff θ_1 and θ_2 are strongly compatible and $i \neq j$.*

Proposition 3 *Let C and D be clauses. Then $C\theta \subseteq D$ with $\theta = \theta_1 \ldots \theta_{|C|}$, iff there is a clique $\{(\theta_1, 1), \ldots, (\theta_{|C|}, |C|)\}$ of size $|C|$ in the substitution graph of C and D.*

Proof 1 *"$\Rightarrow$" Let $C\theta \subseteq D$. Then each literal l_i of C is embedded in D, i.e., we can split θ into $\theta_1 \dots \theta_{|C|}$, such that $l_i \in C$, $l_i\theta_i \in D$. The θ_i are clearly strongly compatible because it $C\theta \subseteq D$, then no variable can be assigned two different terms in θ. Then $\{(\theta_1, 1), \dots, (\theta_{|C|}, |C|)\}$ is a clique in the substitution graph due to def 12. Furthermore, there can exist no clique of size $> |C|$, because matching substitution for the same literal of C are not adjacent (see def. 12) and there are $|C|$ literals in C only.*

"$\Leftarrow$" Let $\{(\theta_1, 1), \dots, (\theta_n, n)\}$ be the nodes of a clique in the substitution graph. Due to def. 12 the θ_i are mutually strongly compatible. If (θ_i, i) and (θ_j, j) are in the clique, then $i \neq j$, otherwise the nodes had got no edge (see def 12). As there are n matching substitutions for n different literals of C and $n = |C|$, each literal of C is embedded into some literal of D, hence for $\theta = \theta_1 \dots \theta_n$ $C\theta \subseteq D$ holds.

We showed that $C \vdash_\theta D$, iff the substitution graph of C and D contains a clique of size $|C|$. The number of nodes of the substitution graph is $|C||D|$, but we will now discuss an algorithm that needs to search less than $|D|^{|C|}$ nodes, rather than $(|C||D|)^{|C|}$ which would be required to search for a clique of size $|C|$ in a graph of size $|C||D|$.

4.2 Searching for a clique

The following algorithm searches for a clique of size k in a graph, it reflects the basic scheme of most known clique algorithms. $clique(V, E, k)$ returns 'yes', if the graph that is defined by V and E contains a clique of size k, or 'no', otherwise. $neighbors_E(v) = \{v_i | (v, v_i) \in E\}$ is the set of neighbors of v with respect to E.

Algorithm A: $clique(V, E, k) =$

1. If $k = 0$, return 'yes'.
2. Let $v \in V$ be any node
3. If $clique(V \cap neighbours_E(v), E, k-1)$, return 'yes'.
4. Else if $clique(V \setminus \{v\}, E, k)$, return 'yes'.
5. Else return 'no'.

Essentially, we pick the first node, search its neighbors for a clique including it, and, on failure, search for a clique without it. This ensures that every possible clique is found (the split is obviously complete), and that each clique is found only once (a node that is dropped in step 4 is not picked up again). The algorithm terminates, since in each recursive call either k is decremented, or the set of nodes is shrunk (provided there are no reflexive edges, which we assume). Hence, the worst case time complexity of this algorithm is equal to the number of k-elementary subsets of an n elementary set (if $n = |V|$ is the size of the graph we are searching), which is $O(\binom{n}{k})$. In more detail, on the top level the algorithm can choose n nodes, on the next level $n-1$, on the kth level $n-k+1$, no k-elementary subset is enumerated twice, hence, at most $\frac{n(n-1)\dots(n-k+1)}{k!} = \binom{n}{k}$ k-elementary subsets of nodes are enumerated. A more common upper bound of $\binom{O(n}{k)}$ is $O(n^k)$. The worst case of this algorithm is a graph, that is a clique by itself. Here, every k-elementary subset can in fact be enumerated as a clique of size k. Now, how much search is required to check the substitution graph, which possesses $|C||D|$ nodes, for a clique of size $|C|$? As we already pointed out, there is no edge between two matching substitutions that match the same literal in C to different literals in D. Therefore, on the top level the algorithm can choose one of $|C||D|$ nodes, but on the next level only $(|C|-1)|D|$ neighbors of that node remain while on the last level $k = |C|$ there are $(|C|-k+1)|D| = 1|D|$ choices left. Again, no $|C|$-elementary subset is enumerated twice, resulting in a search space of size

$$\frac{|C||D|(|C|-1)|D|\dots|D|}{|C|!} \tag{1}$$

$$= \frac{|C|(|C|-1)\ldots 1|D|^{|C|}}{|C|!} \quad (2)$$

$$= |D|^{|C|} \quad (3)$$

By now, mapping the subsumption problem to the clique problem left the size of the search space unchanged. We can, however, exclude a fair amount of search space. Carraghan and Pardalos (1990) point out, that a recursive descent (*i.e.,* computation of the clique of $V \cap neighbors_E(v)$) is unnecessary, if the size of the best clique found so far is greater or equal to the current depth which is equal to the number of nodes picked so far, plus the number of remaining nodes in $V \cap neighbors_E(v)$. Rather than to the size of the best clique found so far, we can refer to the size k we are actually interested in, and stop the search, if the current V contains less than the current k nodes.

Algorithm B: $clique(V, E, k) =$

1. If $k = 0$, return 'yes'.
2. If $|V| < k$ return 'no'.
3. Let $v \in V$ be any node
4. If $clique(V \cap neighbours_E(v), E, k-1)$, return 'yes'.
5. Else if $clique(V \setminus \{v\}, E, k)$, return 'yes'.
6. Else return 'no'.

Considering additional knowledge about the subsumption problem, we can identify another class of regions of the search space that cannot contain solutions. Again, recall that no globally consistent substitution can contain two matching substitutions, that match the same literal in C to different literals in D (each variable x in C has to match *one* $x\theta$ in D). Hence, no clique can contain two nodes augmented with equal numbers of originating literals, because it directly contradicts the definition of edges of the substitution graph. We can therefore stop the search, if the number of different augmented numbers in the current V is less than the number of nodes that are missing to make up a clique of the desired size.

Algorithm C: $clique(V, E, k) =$

1. If $k = 0$, return 'yes'.
2. If $|\{i|(\theta_j, i) \in V\}| < k$ return 'no'.
3. Let $v \in V$ be any node
4. If $clique(V \cap neighbours_E(v), E, k-1)$, return 'yes'.
5. Else if $clique(V \setminus \{v\}, E, k)$, return 'yes'.
6. Else return 'no'.

We will now compare this algorithm to the algorithm proposed by Kim and Cho (1992).

Algorithm Kim and Cho:

1. Generate the set of matching substitutions
2. Delete those substitutions, that are strongly compatible with less than $|C|-1$ different substitutions
3. enumerate the cartesian product of the remaining substitutions of dimension n and check, if there is a strongly compatible n-tuple.

This notation of the algorithm is essentially identical to the algorithm in (Kim & Cho, 1992) that is expressed in terms of bit vectors. Although this algorithm does not refer to the clique problem, we will explain that the search space set up by the cartesian product of the remaining matching substitutions is a superset of the search space the clique has to be found in. Since the set of matching substitutions sets up the vertices of the substitution graph and the compatibility relation sets up the edges, any n-tuple of pairwise strongly compatible substitutions sets up a clique in the substitution graph. Hence point (3) of the Kim and Cho algorithm performs the search for a clique of size n. Point (2) excludes substitutions with less than $|C|-1$ compatible substitutions, or, in terms of substitution graphs, excludes nodes with an output degree of less than $|C|-1$. Our rewritten pruning rule (point 2 of algorithm C) in turn does not expand a node, if the current set of remaining nodes contains indices referring to less than k literals, where k is initially equal to $|C|$. The rule fires when called from the top level, iff $|\{i|(\theta_j, i) \in V \cap neighbors(v)\}| < |C| - 1$ – *i.e.,* if n is compatible with less than $|C|-1$ substitutions referring to different literals of C which is of course a subset of all compatible substitutions of v. As $neighbours(v)$ is the degree of the vertex (θ_j, i) in the substitution graph, rule 2 of algorithm C is more general than the Kim and Cho rule, in the sense that if the Kim and Cho rule fires, rule 2 fires as well, but not vice versa. Furthermore, the rule may fire on depths > 1, pruning additional nodes.

Kim and Cho additionally propose a second pruning strategy: If two incompatible matching substitutions possess an equal set of adjacent nodes, then one of them can be removed. This directly corresponds to a simple symmetry detection in clique search and may increase the performance of the algorithm: If two non-adjacent nodes share the same set of adjacent nodes, there are at least two cliques of identical size containing exactly one of these nodes, and it is irrelevant which one of them is chosen.

4.3 Fast subsumption

We are now able to join the approaches elaborated in sections 3 and 4 to a fast algorithm for testing θ-subsumption.

Algorithm Fast θ-subsumption $\vdash_\theta (C, D)$:

1. Match as many literals of C deterministically to literals of D. Substitute C with the substitution found. If any literal of C does not match any literal of D then decide $C \not\vdash_\theta D$.
2. Match as many literals of C context based deterministically to literals of D. Substitute C with the substitution found. If any literal of C does not match any literal of D then decide $C \not\vdash_\theta D$.
3. Set up the substitution graph $(V_{C,D}, E_{C,D})$. Delete all nodes $\mu \in V_{C,D}$ with $l_C\mu = l_D$ and $con(l_C, d, C) \not\subseteq con(l_D, d, D)$. Use algorithm C to test $(V_{C,D}, E_{C,D})$ for a clique of size $|C|$.

This algorithm combines the benefits of both approaches. Clauses that can be tested context based deterministically are tested in polynomial time without any search, in other cases the algorithm matches as many literals deterministically as possible, and sets up the remaining substitution graph. It follows from proposition 1 that we can delete nodes that match literals of C to literals in D which do not possess at least the same context, because no such substitution can be contained in any globally consistent solution. We search the remaining graph for a clique, using the algorithm explained in Section 4 to exclude as much of the search space as possible.

5. Empirical results

Our experiments are based on a set of large graphs that encode finite element meshs of construction parts. This data set is well known in the ILP community (Bolsak & Muggleton, 1992), where it is used as a benchmark learning problem; The classification problem is to predict the optimal discretization of edges. To obtain different clauses of arbitrary size, we included only those nodes and edges that have only a certain distance from a randomly drawn starting node. We computed the least general generalization (Plotkin, 1970) of pairs of such clauses and drew a random subset of the body literals to adjust the clause size more precisely. The D clauses were generated using a fixed variable depth They are of approximately the same size (approximately 130 literals). We varied the size of C and tested for $C\theta \subseteq D$. For each curve we used about 5,000 to 10,000 subsumption tests in the positive and up to 25,000 tests in the negative case, taking 140 days of computation time on Sparc20 workstations in total. The matches-deterministic, and the context based algorithms, invoke the plain prolog-like matching algorithm after the candidate sets are reduced by the context inclusion criterion that searches the remaining space. The combination of the graph context and clique algorithm maximally reduces the set of matching candidates and invokes the clique algorithm to search the remaining substitution space. All algorithms are implemented in "C" and the experiments were performed on sparc stations.

We examined mainly two questions: (1) How does the maximum clique approach compare to a plain subsumption algorithm and (2) How does the context based approach compare to the deterministic match, especially for negative examples, which are the major problem of the prolog-like algorithms.

The main problem we were facing is the high variance of the measured time. While 95% of the problems can be solved in only a fraction of the average time, very rare cases occur that require several hours – up to several days – to be solved. As we were not able to observe a sufficiently large number of these rare and very expensive cases – which would have required several hundred days – our curves for the positive case are fairly noisy. Yet, we are able to state significant results covering 95% of the problems, after we omitted those 5% of cases with the largest deviation from the mean value. Fig. 1 shows the time results in the positive case for 100%, fig. 2 for 95% of the observed problems. The strong similarity between the two diagrams indicates that the exclusion of 5% extrema successfully eliminates the noise but does not influence the obtained results. In contrast to Kietz and Lübbe (1994) who obtained an improvement of performance using deterministic subsumption (based on an artificial data set), we cannot confirm that deterministic subsumption yields an improvement on the mesh design data set in the positive case. The context based algorithms clearly improve the performance, the curve is shifted by about 10 literals[1]. Although the difference in performance between a depth of 1 and 2 is rather small, the graph context based approach at a depth of 2 yields the best results. While the clique based subsumption shows an impressive behavior, the combination of the graph context and the clique approach yields an even better result. The mean time for 50 literals is only 1 second, while this problem is completely intractable for the plain subsumption algorithm.

Figure 3 shows the results for the negative case ($C\theta \not\subseteq D$). Since the tests are much faster in the negative case, we were able to perform up to 25,000 tests, hence the curves are less noisy. The deterministic test is significantly faster than the plain subsumption in the negative case, in fact more than 95% of all problems are solved in less than 0.01 second. This happens when a literal is found that does not match any literal of the other clause; otherwise backtracking has to be performed. The most expensive observed test with the deterministic matching algorithm required 34 hours, the most expensive graph context based test, by contrast, required 4 seconds. The most expensive test done by the combination of the context and clique based algorithm, based on 25,000 observations, took 0.4 seconds.

1. The curves labeled 'literal context' show the behavior of a modified context based algorithm discussed in (Scheffer et al., 1996)

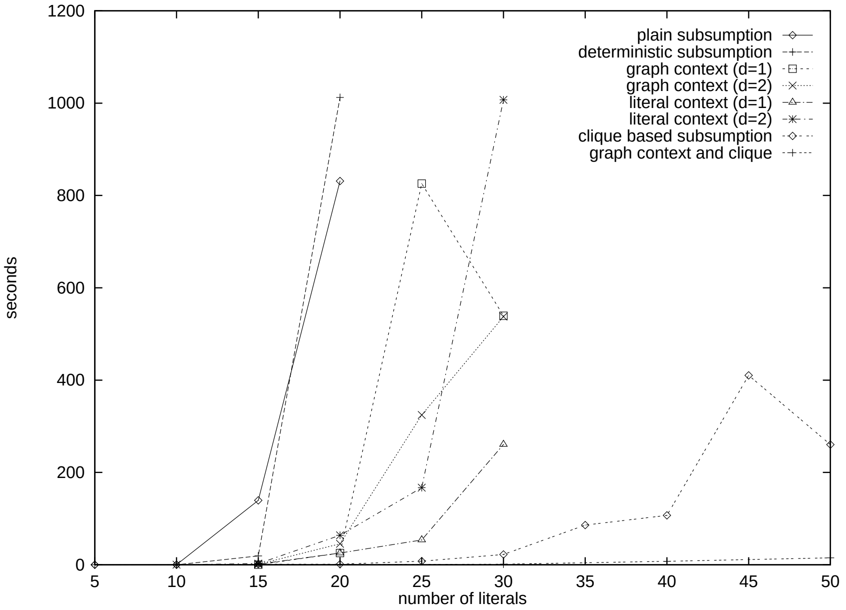

Figure 1: Average time for 100% of the positive samples

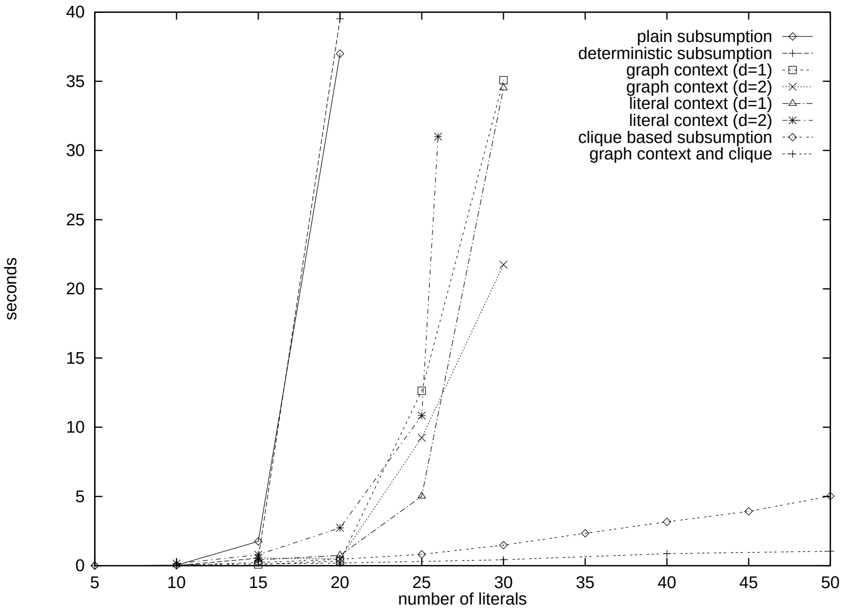

Figure 2: Average time for 95% of the positive samples

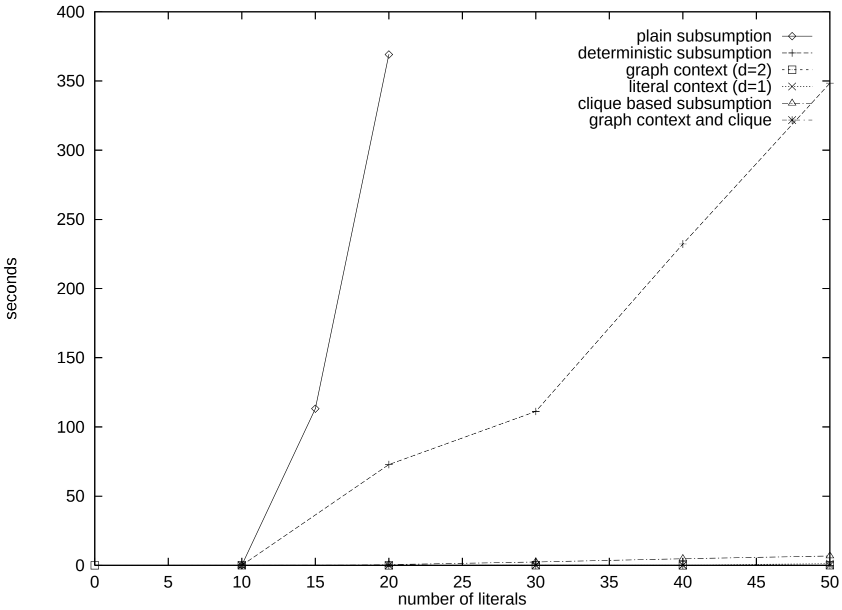

Figure 3: Average time for negative samples

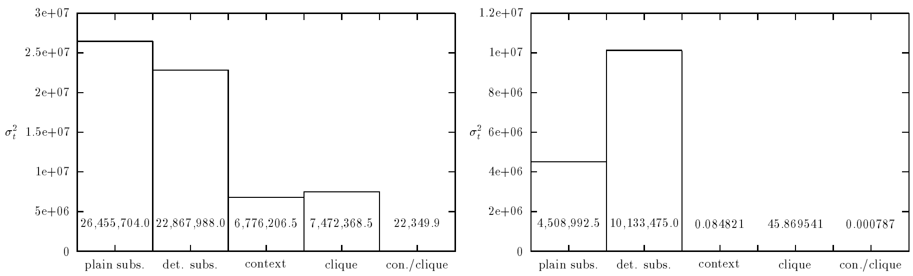

Figure 4: Variance for positive and negative samples

Figure 4 shows the variances $\frac{1}{n}\sum(t_i - \mu_{s_i})^2$, where t_i is the observed time and μ_{s_i} the mean time for a clause size of s_i literals, for the studied algorithms in the positive and negative case respectively. In the positive case, the variance is very high in general – *i.e.,* most tests require only a fraction of the mean time, the mean value is strongly influenced by a small number of expensive tests. Both, the context approach and the pruning strategy of the clique algorithm reduce the number of these expensive tests. The combination of these approaches reduces the variance by a factor of 1000. In the negative case the differences are even larger. The deterministic subsumption shows a very high variance as expected, because more than 95% of all problems are solved instantly. As the candidate elimination procedure leaves only a very small search space (if any), and the pruning strategy works very reliable, the combined algorithm solves all observed problems very fast and shows a very low variance in the time required.

6. Conclusion and further research

We proposed two approaches to coping with the complexity of the θ-subsumption problem. One approach is context based candidate elimination. By not matching variables to terms that are not "surrounded" by at least the same literals up to an arbitrary depth, in many cases the subsumption becomes deterministic and backtracking becomes obsolete. The set of clauses that this algorithm matches in polynomial time is a superset of the determinate clauses. The second part of this paper particularly addresses cases in which the search space cannot be reduced to a single element and identifies additional parts of the search space that cannot contain globally consistent solutions.

Our empirical results are based on the finite element mesh design data set which consists of graphs that represent finite element meshs of construction parts. In the negative case ($C \not\vdash_\theta D$) the context based algorithm is in nearly any case able to decide non-subsumption without search. In the remaining cases, the search space left is very small. In the positive case, context based candidate elimination reduces the search space by an equivalent of 10 literals. The clique based algorithm, in particular when combined with candidate elimination, reduces the search space to a small fraction of its original size.

The proposed combination of the context and the clique based subsumption algorithm can in turn be combined with the k-local match (Gottlob & Leitsch, 1985; Kietz & Lübbe, 1994). If C contains classes of literals such that there are no common variables in different classes, each class can be matched independently and the complexity grows exponentially with the size of the largest local only. Each class can be matched using the presented algorithm.

The efficiency of the θ-subsumption test is important to the performance of theorem provers and crucial to the performance of ILP learning algorithms. This is of special importance to generalization based learning algorithms that usually generate larger clauses and have to test generalized clauses for consistency with respect to a huge set of samples. Hence, our future work will focus on graph based machine learning algorithms, that make use of the presented efficient matching algorithms.

References

Bolsak, B., & Muggleton, S. (1992). The application of inductive logic programming to finite-element mesh design. In *Inductive Logic Programming* London. Academic Press.

Carraghan, R., & Pardalos, P. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters*, *9*, 375–382.

DeRaedt, L., & Bruynooghe, M. (1993). A theory of clausal discovery. In *Proc. Workshop on ILP*.

Dzeroski, S., Muggleton, S., & Russel, S. (1992). Pac-learnability of determinate logic programs. In *Proc. 5th ACM Workshop on Computational Learning Theory*, pp. 128–135.

Eisinger, N. (1981). Subsumption and connection graphs. In *Proc. IJCAI*.

Feige, U., Goldwasser, S., Lovasz, L., Safra, S., & Szegedy, M. (1991). Approximating the maxclique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Sci.*

Geibel, P., & Wysotzki, F. (1996). Learning relational concepts with decision trees. In *Proc. ICML*.

Gibbons, L., Hearn, D., & Pardalos, P. (1996). A continuous based heuristic for the maximum clique problem. In *Clique, Graph Coloring and Satisfiability: Second DIMACS Implementation Challenge*.

Gottlob, G. (1987). Subsumption and implication. *Information Processing Letters*, *24*, 109–111.

Gottlob, G., & Leitsch, A. (1985). On the efficiency of subsumption algorithms. *J. ACM*, *32*(2), 280–295.

Johnson, D. S., & Trick, M. A. (Eds.). (1996). *Clique, Graph Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS series.

Kapur, D., & Narendran, P. (1986). NP-completeness of the set unification and matching problems. In *Proc. 8th International Conference on Automated Deduction*.

Kietz, J.-U., & Lübbe, M. (1994). An efficient subsumption algorithm for inductive logic programming. In *Proc. International Conference on Machine Learning*.

Kim, B. M., & Cho, J. W. (1992). A new subsumption method in the connection graph proof procedure. *Theoretical Computer Science*, *103*, 283–309.

Kowalski, R. (1975). A proof procedure using connection graphs. *J. ACM*, *22*(4), 572–595.

Loveland, D. W. (1978). *Automated theorem proving: A logical basis.* Elsevier, North Holland.

Muggleton, S. (1993). Inverting implication. *Artificial Intelligence Journal*.

Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *Proc. 1st Conf. on Algorithmic Learning Theory*, pp. 368–381.

Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence*, Vol. 5, pp. 153–163.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *J. ACM*, *12*(1), 23–41.

Scheffer, T., Herbrich, R., & Wysotzki, F. (1996). Efficient theta-subsumption based on graph algorithms. In *Proc. Int. Workshop on Inductive Logic Programming*.

Sickel, S. (1976). A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, *C-25*(8), 823–835.

Socher, R. (1988). A subsumption algorithm based on characteristic matrices. In *Proc. 9th Int. Conf. on Automated Deduction*.

Tinhofer, G. (1976). Zum algorithmischen Nachweis der Isomorphie von endlichen Graphen. In Noltemeier, H. (Ed.), *Graphen, Algorithmen, Datenstrukturen. 2. Fachtagung über Graphentheoretische Konzepte der Informatik*. Carl Hanser Verlag.

Unger, S., & Wysotzki, F. (1981). *Lernfähige Klassifizierungssysteme*. Akademie Verlag Berlin.

van der Laag, P., & Nienhuys-Cheng, S. (1993). Subsumption and refinement in model inference. In *Machine Learning: ECML*.

Weisfeiler, B. (1976). *On Construction and Identification of Graphs*. No. 558 in Lecture Notes in Mathematics. Springer Verlag, Berlin.

Wysotzki, F., Selbig, J., & Kolbe, W. (1981). Concept learning by structured examples – an algebraic approach. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*.