Efficient θ-Subsumption based on Graph Algorithms*

Tobias Scheffer, Ralf Herbrich and Fritz Wysotzki

Technische Universität Berlin, Artificial Intelligence Research Group, Sekr. FR 5-8, Franklinstr. 28/29, D-10587 Berlin, email: scheffer@cs.tu-berlin.de

Abstract. The θ-subsumption problem is crucial to the efficiency of ILP learning systems. We discuss two θ-subsumption algorithms based on strategies for preselecting suitable matching literals. The class of clauses, for which subsumption becomes polynomial, is a superset of the deterministic clauses. We further map the general problem of θ-subsumption to a certain problem of finding a clique of fixed size in a graph, and in return show that a specialization of the pruning strategy of the Carraghan and Pardalos clique algorithm provides a dramatic reduction of the subsumption search space. We also present empirical results for the mesh design data set.

1 Introduction

θ-subsumption [Rob65] is a correct but incomplete, 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 $C\theta \subseteq D$.

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. [MF90, vdLNC93, DB93]. Especially the consistency test, i.e. the test if a newly generalized clause covers negative samples, requires a large amount of subsumption tests. Hence, efficient subsumption algorithms that do not come along with restrictions of the hypothesis language, are an important contribution to ILP.

θ-subsumption of two clauses is NP-complete in general [KN86], even if the second clause is fixed [KL94]; the NP-completeness results from the ambiguity of variable identification. As subsumption is performed very often in ILP, the efficiency is crucial to the power of ILP learners; many approaches to speeding up subsumption were studied.

If in a clause a literal can be found, that matches exactly one literal of the other clause, this literal can be matched *deterministically* [DMR92, KL94] and no backtracking needs to be done. Thus, the complexity grows exponentially with the number of remaining, non-deterministic literals. In this paper, we will propose to reduce the number of candidates for each literal using context information, such that for some literals only one candidate remains, and can be matched deterministically.

* Proc. International Workshop on Inductive Logic Programming, 1996

For the similar problem of graph isomorphism, there are several approaches, [Tin76, Wei76, WSK81, UW81, GW96a, GW96b], to reducing matching candidates using context information. We will adapt a very general approach to the problem of subsumption and show, that characteristic matrices [Soc88] are a special case of this approach. We will present a second approach as well and characterize the set of clauses, that can be subsumed in polynomial time by this algorithm.

Eisinger [Eis81] introduces *S-links* into the framework of the connection graph resolution proof procedure [Kow75, Sic76]. 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 [KC92] 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. [FGL$^+$91], yet much effort has been spent in the search for algorithms that behave efficient in the average, e.g. [JT96, CP90, GHP96]. We will show, that subsumption can be tested by finding a clique of size n, and we will show that a specialization of the Carraghan and Pardalos pruning strategy [CP90] strongly reduces the search space, we will show that this space is smaller than the cartesian product space proposed by Kim and Cho [KC92].

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 [GL85, KL94].

In sections 3 and 4, we describe two alternative context based algorithms, while in section 5 we describe our clique based approach. These sections may be read independently.

2 The θ-subsumption problem

θ-subsumption [Rob65, Plo70] is an approximation of the 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 [Lov78], 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, if C is self-resolving (recursive) or if D is tautological [Got87]. If tautologies and self-resolution are excluded, then $C \vdash_\theta D \Leftrightarrow C \models D$ [Got87, Mug93, KL94].

The θ-subsumption problem is NP-complete in general [KN86]. The worst case time complexity is $O(vars(D)^{vars(C)})$, or $O(|D|^{|C|})$.

Definition 1. A substitution is a mapping from variables to terms. We denote substitutions $\theta = \{x_1 \to t_1, \ldots, x_n \to t_n\}$.

Definition 2. 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. 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 for each literal there is exactly one possible match that is consistent with the previously matched literals [MF90] or, more generally, if there is an ordering of literals, such that at each step for each literal there is exactly one match that is consistent with the previously matched literals [KL94].

Definition 4 deterministic subsumption. Let $C = c_0 \leftarrow \{c_i\}$ 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 $c_1, \ldots, c_n$ of the c_i, such that for all i, $1 \leq i \leq n$ there exists exactly one θ_i, such that $\{c_1, \ldots c_i\}\theta_0 \ldots \theta_i \subseteq D$.

As Kietz and Lübbe point out [KL94], the definition of determinate subsumption between two clauses is identical to the definition with respect to one clause, a set of background literals and an example [MF90]: if an observation e follows from a clause C and a set of background literals l_i, then $C \vdash e \leftarrow \{l_1, \ldots, l_n\}$.

$C \vdash_{\theta DET} D$ can be tested with at most $O(|C|^2 \cdot |D|)$ unification attempts [KL94] 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 the condition is very strict. In our experiments with the mesh design data set we learned, that almost no literal at all could be matched deterministically, unless the data set was especially prepared (see section 6). Yet, 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 two algorithms, the condition holds more often and the set of clauses, to be subsumed deterministically, grows.

3 Reduction of matching candidates using graph context

Wysotzki [WSK81, UW81] proposes an approach to reducing the number of matching candidates for the graph isomorphism problem that is based on results of Weisfeiler [Wei76] and Tinhofer [Tin76]. It was applied to obtaining attribute-value data for machine learning of graph classification rules [GW96a, GW96b]. The approach is based on the idea, that nodes may be matched to those nodes only, 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 yields the following:

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

It is important to get an understanding of this procedure: 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, each summand representing one path. In general, A^n_{ij} enumerates all paths of length n. We shall 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 shall now 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 for the isomorphism test 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, and the comparison of paths is $O(paths^2) = O(|V|^{2k})$. But for not too large a k this is feasible and it showed, that almost any isomorphism test can be performed in polynomial time [Wei76, WSK81, UW81]. It is known, that two nodes cannot match if their context paths do not match, but it is unknown, if a match of the context implies a 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 mind two main differences: The concept of a path does not cover more than binary relations and we do not want to decide the identity of clauses (up to substitution), but the inclusion of clauses (up to substitution) instead. The translation is based on the fact, that for each occurrence of a variable x_1 in C there must be a corresponding occurrence of $x_1\theta$ in D. We define the occurrence graph to denote the occurrence of identical variables. We firstly focus on datalog clauses, i.e. clauses with terms restricted to variables or constant symbols.

Definition 5 occurrence graph. (C, E_C) is the occurrence graph of a datalog clause C with if $(l_i, l_j, \pi_i \leftrightarrow \pi_j) \in E_C$ 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 $\pi_i \leftrightarrow \pi_j$, where π_i and π_j are argument position in which occurrences of the same variable are found.

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

Example 2. Let $C = r(x_1, x_2), r(x_2, x_3), q(x_3)$. The occurrence graph contains only two edges: $(r(x_1, x_2), r(x_2, x_3), 2 \leftrightarrow 1)$ and $(r(x_2, x_3), q(x_3), 2 \leftrightarrow 1)$, denoting 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 a variable on position 2 or $r(x_2, x_3)$ that also occurs on position 1 of $q(x_3)$. The graph context of literal $r(x_1, x_2)$ at depth 1 contains only the term $r \cdot 2 \leftrightarrow 1 \cdot r$. At depth 2 the context contains two paths: $r \cdot 2 \leftrightarrow 1 \cdot r \cdot 2 \leftrightarrow 1 \cdot q$ and $r \cdot 2 \leftrightarrow 1 \cdot r \cdot 1 \leftrightarrow 2 \cdot r$. The first path corresponds to the literal sequence $r(x_1, x_2) - r(x_2, x_3) - q(x_3)$, the second to the sequence $r(x_1, x_2) - r(x_2, x_3) - r(x_1, x_2)$.

Proposition 7. *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 $con_{gra}(l_1, d, C) \not\subseteq con_{gra}(l_2, d, D)$, then there is no θ, such that $C\mu\theta \subseteq D$.*

That is, a literal must not be matched against another literal, if its context cannot be embedded in the other literal's context.

Outline of proof 1 *Let C and D be clauses, let $con_{gra}(l_1, d, C)$ and $con_{gra}(l_2, d, D)$ be the context of a literal from C and D respectively, such that $con_{gra}(l_1, d, C) \not\subseteq con_{gra}(l_2, d, D)$. 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 $p_a \cdot \pi_a \leftrightarrow \pi_b \cdot p_b$ 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 [Soc88] 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 in turn contains a path for each literal in which a common variable occurs, consisting of the predicate name and a term $i \leftrightarrow k$, 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, 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, a literal in C can be matched with (the matching candidates). 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, then $C\theta \not\subseteq D$, and if there is a literal that has exactly one candidate, then we need not perform backtracking for this literal, which is the idea of deterministic subsumption. In these cases, $C \subseteq D$ can be tested in $O(|C|^2 \cdot |D| \cdot 2^{2d})$, because $|C|^2 \cdot |D|$ is the complexity of deterministic subsumption [KL94] and we need to compare $O(2^d)$ paths to test for $con_{gra}(l_i) \subseteq con_{gra}(l_j)$, where d is the fixed depth.

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, nor are there any locals. At depth 1, the context of the first literal $r(x_1, x_2)$ only contains the path $r \cdot 2 \leftrightarrow 1 \cdot r$ (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 $r \cdot 1 \leftrightarrow 2 \cdot r$. The context of $r(y_1, y_2)$ is $\{r \cdot 2 \leftrightarrow 1 \cdot r, r \cdot 1 \leftrightarrow 1 \cdot r\}$, the context of $r(y_2, y_3)$ is $\{r \cdot 1 \leftrightarrow 2 \cdot r, r \cdot 2 \leftrightarrow 2 \cdot r\}$ and of $r(y_1, y_3)$ is $\{r \cdot 1 \leftrightarrow 1 \cdot r, r \cdot 2 \leftrightarrow 2 \cdot r\}$. 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 non-trivial terms, then we compute a datalog clause C' according to [Soc88] and generate the context w.r.t 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 Reduction of matching candidates using the literal context

In this section, we propose another pruning strategy that reduces the number of matching candidates, that is based on the principle, that identical variables in one clause have to be matched on identical variables in the other clause. We define the literal graph in which literals with common variables are adjacent, but in contrast to the occurrence graph we omit the argument positions.

Definition 8 literal graph. (C, E_C) is the literal graph of a clause C, if and only if $(l_i, l_j) \in E_C$ iff there is a variable x occurring in both, l_i and l_j.

The literal context at a depth d of a literal includes all those literals, that can be reached via a path of length d.

Definition 9 literal context. The context at depth d of a literal $l \in C$ is the clause $con_{lit}(l, C, d)$, that contains exactly those literals, that can be reached via a path of length at most d in the literal graph of C.

The size of the context is growing exponentially with the depth but is limited by the size of the clause C. We write $con_{lit}(l, C, d, k)$ to limit the size to a fixed k, i.e. $con_{lit}(l, C, d, k)$ is a random subset of $con_{lit}(l, C, d)$, of size k.

A literal $l_1 \in C$ must not be matched against a literal $l_2 \in D$, if the context of l_1 cannot be embedded in the context of l_2.

Proposition 10. *Let C and D be clauses and $l_1 \in C$ and $l_2 \in D$ be literals, such that $l_1\mu = l_2$. If there is no θ, such that $con_{lit}(l_1, C, d)\mu\theta \subseteq con_{lit}(l_2, D, d)$ then there is no θ', such that $C\mu\theta' \subseteq D$.*

Outline of proof 2 *Let x be a variable occurring in more than literal of C. These literals only match literals of D containing identical variables $x\theta$ at the corresponding places. This implies that any set of literals of C connected by sequence of pairs of identical variables only matches a set of literals with identical variables in the corresponding pairs of literals. If there is a literal l_j of C in the context of a literal l_i and l_i matches a literal l' of D by a literal match μ, then due to def. 9 it is connected via a chain of variable occurrences and it only matches a literal in the context of $l_i\mu$. If there is no such literal, there can be no global match containing μ.*

We will now focus on the complexity of a match based on the literal context and we will characterize the set of clauses, that can be matched in polynomial time.

Clearly, the size of the context is growing exponentially with the depth, but it is restricted to the size of the clause and furthermore can be restricted to an arbitrary size k. Note that, if $C\theta \subseteq D$ is to be tested, the size of the context of the literals of C can be restricted to an arbitrary k, while the size of the context of the D literals has to be computed to at least the same depth and cannot be restricted to a random subset of size k, because if $A \subset B$, then any subset of A is a subset of B as well, but need not be a subset of a random subset of B.

Definition 11 Generalized 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 $con(d,k)$-deterministically subsumes D by $\theta = \mu_0 \ldots \mu_n$, written $C \vdash_{\theta dkDET} D$, iff there exists an ordering $l_1, \ldots, l_n$, such that for all i, $1 \leq i \leq |C|$, there exists exactly one μ, such that there is a $l' \in D$ and $l_i \mu = l'$ and $con_{lit}(l_i, C, d, k)\mu\theta'_i \subseteq con_{lit}(l', D, d)$.

Clearly, this is a generalization of the determinacy concept, because for the context depth of 0, $con_{lit}(l_i, C, d, k)$ is the empty set and the definition becomes identical to the definition of the 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 more often there exists exactly one remaining matching candidate.

We know, that the deterministic match can be tested in $O(|C|^2 \cdot |D|)$, we furthermore know, that in general subsumption can be tested in $O(|D|^{|C|})$. Thus, the context inclusion can be tested in $O(|D|^k)$, because the context of the literal from D is restricted by the size of the whole clause, while we can restrict the context of the literal from C to any arbitrary k. Since we only have to modify the deterministic matching algorithm in a way that it does not only test the literals for matching substitutions but tests for context inclusion instead, the $con(d,k)$-deterministic match can be tested in $O(|D|^k \cdot |C|^2 \cdot |D|)$.

Example 4. Let $C = r(x_1, x_2), r(x_2, x_3)$ and $D = r(y_1, y_2), r(y_2, y_3), r(y_4, y_3)$. We want to test if $C\theta \subseteq D$. Note again, that the clauses cannot be matched deterministically, nor are there any locals. At depth 1, the context of $r(x_1, x_2)$ contains $r(x_2, x_3)$ and vice versa. The context of $r(y_1, y_2)$ contains $r(y_2, y_3)$, the context of $r(y_2, y_3)$ is $\{r(y_1, y_2), r(y_4, y_3)\}$ and the context of $r(y_4, y_3)$ contains $r(y_2, y_3)$. Thus, $r(x_1, x_2)$ matches all literals in D but the context can be embedded in the context of $r(y_1, y_2)$ only; the context of $r(x_2, x_3)$ subsumes the context of $r(y_2, y_3)$. Again we can match the clauses in polynomial time, without any need for backtracking.

5 Clique and the general subsumption problem

In this section we show that the subsumption problem can be mapped to the clique problem. We present a specialization of the Carraghan and Pardalos [CP90] pruning strategy.

Definition 12. A pair (V, E) of vertices and edges with $E \subseteq V \times V$ we call a graph.

Definition 13. A set of nodes $C \subseteq V$ is a *clique* of a graph (V, E), iff $E \supseteq C \times C$, i.e. all nodes are mutually adjacent.

5.1 S-link method of subsumption

Eisinger [Eis81] proposes a subsumption test that is based on selecting a compatible tuple of substitutions.

Definition 14. Two substitutions θ_1 and θ_2 are called *strongly compatible* iff $\theta_1 \cdot \theta_2 = \theta_2 \cdot \theta_1$, i.e. no variable is assigned different terms in θ_1 and θ_2.

Definition 15. $uni(C, l_i, D) = \{\mu | l_i \in C, l_i\mu \in D\}$ is the set of all matching substitutions from a literal l_i in C to to some literal in D.

Proposition 16 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 5. Let $C = \{P(x,y), P(y,z)\}$ and $D = \{P(a,b), P(b,c), Q(d)\}$. Then $uni(C, P(x,y), D) = \{\{x \to a, y \to b\}, \{x \to b, y \to c\}\}$ and $uni(C, P(y,z), D) = \{\{y \to a, z \to b\}, \{y \to b, z \to c\}\}$. The cartesian product of these sets is $\times_{i=1}^{2} uni(C, l_i, D) = \{\{x \to a, y \to b\} \cdot \{y \to a, z \to b\}, \{x \to a, y \to b\} \cdot \{y \to b, z \to c\}, \{x \to b, y \to c\} \cdot \{y \to a, z \to b\}, \{x \to b, y \to c\} \cdot \{y \to b, z \to c\}\}$, of which only $\{x \to a, y \to b\} \cdot \{y \to b, z \to c\} = \{x \to a, y \to b, z \to c\}$ is a strongly compatible substitution.

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 17 substitution graph. Let C and D be clauses and $n = |C|$. Then $(V_{C,D}, E_{C,D})$ is the substitution graph iff $V_{C,D} = \bigcup_{i=1}^{n} (uni(C, l_i, D), i)$ 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 18. *Let C and D be clauses. Then $C\theta \subseteq D$ with $\theta = \theta_1 \cdot \ldots \cdot \theta_n$, iff there is a clique $\{\theta_1, \ldots, \theta_n\}$ of size $|C|$ in the substitution graph of C and D.*

Proof 4 *"$\Rightarrow$" Let $C\theta \subseteq D$. Then each literal l_i of C is embedded in D, i.e. we can split θ into $\theta_1 \cdot \ldots \theta_n$ 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 $(\theta_1, 1), \ldots, (\theta_n, n)$ is a clique in the substitution graph due to def 17. Furthermore, there can exist no clique of size $> n$, because matching substitution for the same literal of C are not adjacent (see def. 17) and there are n literals in C only.*

"$\Leftarrow$" Let $((\theta_1, 1), \ldots, (\theta_n, n))$ be a clique in the substitution graph. Due to def. 17 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 17). 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 $\theta = \theta_1 \cdot \ldots \cdot \theta_n$ $C\theta \subseteq D$ holds.

5.2 Searching for MAXCLIQUE

Carraghan and Pardalos present the following algorithm to determine the largest clique of a graph [CP90]:

1. Start with initial sets *nodes* containing all nodes, *best-clique*, initially empty, and a recursion *depth*, initially 1.
2. For all nodes θ_i in *nodes* repeat
   (a) select a new set $nodes' = nodes \cap neighbors(\theta_i)$
   (b) if $|best\text{-}clique| < depth + |nodes'|$ (and $nodes' \neq \emptyset$), then starting recursively from point (2) with $depth + 1$ and $nodes'$, determine the max clique of $nodes'$.
   (c) If $\{\theta_i\}\cup$ max clique of $nodes'$ is larger than the *best-clique*, save the new clique in *best-clique*.
   (d) remove θ_i *and from nodes* and continue the loop at point 2.
3. return *best-clique*

Example 6.

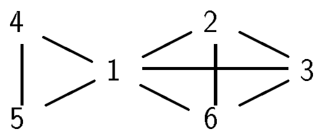

Fig. 1. Graph used in example 6

depth	nodes	θ_i	nodes'	best-clique	comment
1	123456	1	23456	-	
2	23456	2	36	-	
3	36	3	6	-	
4	6	6	-	6	$nodes'$ empty, return
3				36	
2				236	
2	3456	3	6	236	$depth + nodes' = 2 + 1 \leq 3 \rightarrow$ prune
2	456	4	5	236	as above
2	56	5	-	236	$nodes'$ empty, return
1				1236	
1	23456	2	36	1236	$depth + nodes' = 1 + 2 \leq 4 \rightarrow$ prune
1	3456	3	6	1236	$depth + nodes' = 1 + 1 \leq 4 \rightarrow$ prune
1	456	4	5	1236	$depth + nodes' = 1 + 1 \leq 4 \rightarrow$ prune
1	56	5	-	1236	$nodes'$ empty, return 1236.

The important aspect of this algorithm is point 2b. A current node θ_i is expanded, only if the number of nodes expanded so far plus the number of nodes not expanded yet, that are neighbors of the current node, exceeds the size of the best clique found so far. Since all nodes in a clique are mutually adjacent, any clique containing node θ_i cannot include any node that is not a neighbor of θ_i. Obviously, there is no clique containing the current clique and θ_i, that is larger than the current clique plus the neighbors of θ_i plus 1 (namely θ_i itself). If the best clique found so far is even larger, we need not further expand θ_i. This pruning strategy provides an optimal reduction, if the best clique known so far is almost as large as the maximum clique in the graph. For the pruning to become mostly efficient, the nodes should be sorted by decreasing output degree, such that large cliques are most likely to be found early. Note that due to point (2a), $nodes \subseteq neighbours(\theta_1) \cap \ldots \cap neighbours(\theta_{n-1})$.

The Carraghan and Pardalos clique algorithm can be used to decide if the substitution graph contains a clique of size n. But while this algorithm uses the best clique found so far to decide, if the pruning strategy can be applied, we now know that no clique larger than $|C|$ can exist; thus, we can use the full power of the pruning strategy from the beginning. We can rewrite the pruning strategy in order to find cliques of size n:

(2b') if $depth + |nodes'| \geq n$, then starting recursively from point (2) with $depth+1$ and $nodes'$, determine the max clique of $nodes'$.

This rule can still be improved. Remember, that in definition 17 we augmented each substitution with the number of the literal from the left hand side clause. Two nodes of the substitution graph are adjacent, only if their literal numbers are not identical (see def 17 and note, that each literal can be matched to one literal of the right hand side clause only). We can therefore state, that there can be no clique of size $i + j$ which is a superset of a current clique of size i, if the set of remaining nodes does not contain at least j different literal numbers.

(2b") if $depth + |\{i | \exists(\theta_i, i) \in nodes'\}| \geq n$, then starting recursively from point (2) with $depth + 1$ and $nodes'$, determine the max clique of $nodes'$.

We will now compare this algorithm to the algorithm proposed by Kim and Cho [KC92].

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 [KC92], 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 $|C|-1$. The rewritten rule (2b') in turn does not expand a node i, if $depth + |nodes'| < n$, where $nodes' = nodes \cap$neighbors of i. At the top level, where $depth = 1$ and $nodes = V$, both rules are identical. If $depth > 1$ (note that $depth$ is always identical to the number of nodes in the current clique), then obviously $neighbour(\theta_1) \cap \ldots \cap neighbour(\theta_i) \subseteq neighbours(\theta_i)$ (remember, that $nodes' \subseteq neighbour(\theta_1) \cap \ldots \cap neighbour(\theta_i)$). Since $neighbour$ is irreflexive (see def. 17), at least $depth - 1$ nodes (namely $\theta_1, \ldots, \theta_{i-1}$), that are neighbors of θ_i are missing in $nodes'$, i.e. $depth + |nodes'| \leq depth + |neighbour(\theta_1) \cap \ldots \cap neighbour(\theta_i)| \leq neighbours(\theta_i)$. As $neighbours(\theta_i)$ is the degree of the vertex θ_i in the substitution graph, the rewritten Carraghan and Pardalos rule is more general than the Kim and Cho rule, i.e. if the Kim and Cho rule fires, this implies that the Carraghan and Pardalos rule fires as well, but not vice versa.

Kim and Cho additionally propose a second pruning strategy: If two incompatible matching substitutions possess an identical 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, then 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.

6 Empirical results

Our experiments are based on the well known finite element mesh design data set [BM92]. We used the complete and non-determinate data set, provided by the MLnet server at the GMD. To obtain clauses C of any size, we set up clauses that contained a `mesh/2` literal as a head and the set of literals linked to the head via an increasing variable depth as the body. We computed the lgg [Plo70] of two such clauses and drew a random subset of the body literals to more precisely adjust the clause size. 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. The combination of the graph context and clique algorithm first 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 do the context based approaches 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 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 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. 2 shows the time results in the positive case for 100%, fig. 3 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 [KL94] 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. Although the difference between the algorithms 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 produces 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 4 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 occurs, if a literal is found, that does not match any literal of the other clause; otherwise backtracking has to be performed. The most expensive test observed took 34 hours, in contrast the most expensive graph context based test took 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.

Figure 5 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,

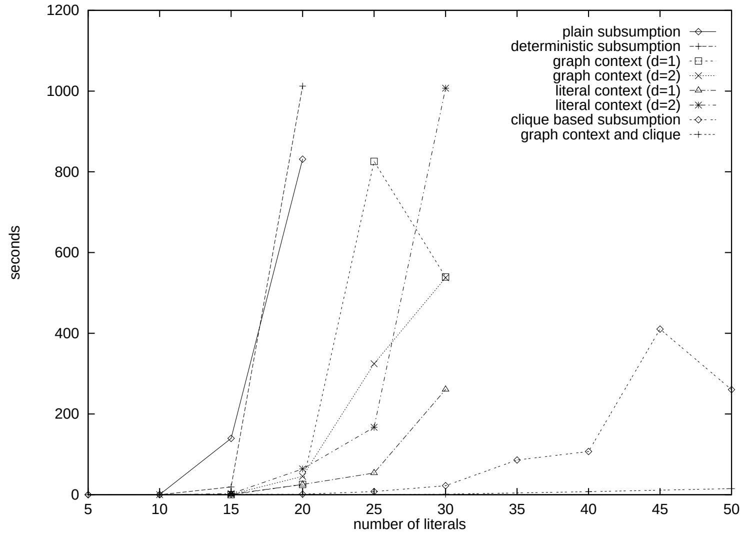

Fig. 2. Average time for 100% of the positive samples

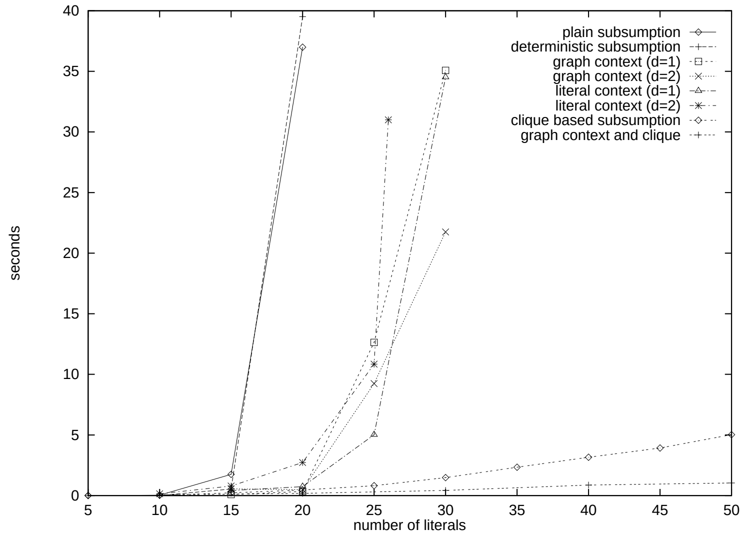

Fig. 3. Average time for 95% of the positive samples

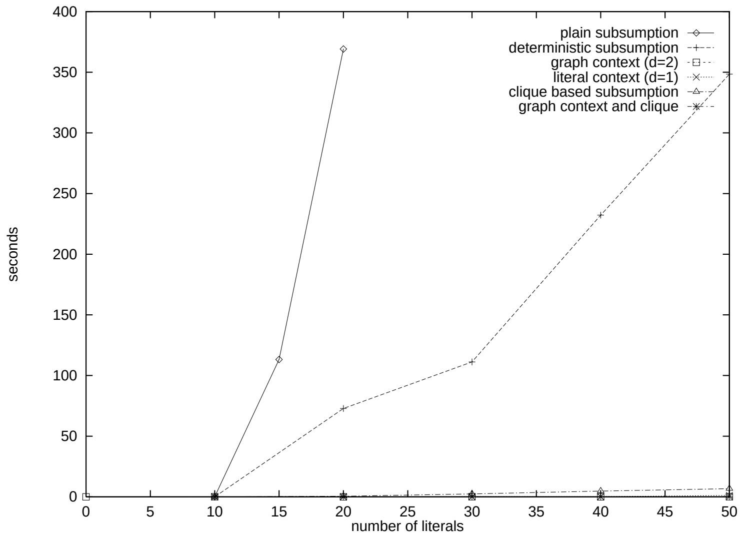

Fig. 4. Average time for negative samples

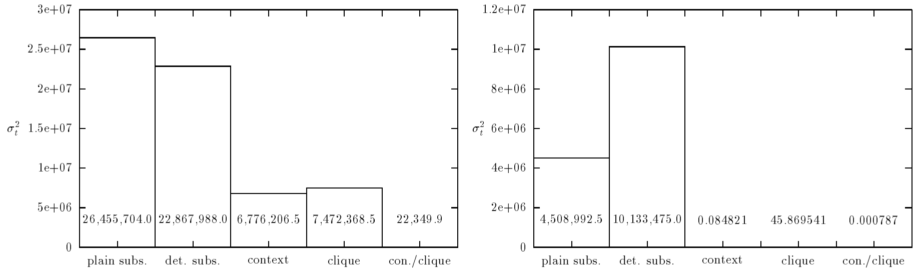

Fig. 5. Variance for positive and negative samples

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.

7 Conclusion and further research

We proposed two approaches to coping with the complexity of the θ-subsumption problem. One approach is a mapping of the subsumption problem to the problem of the maximum clique and the use of a strongly specialized version of the

Carraghan and Pardalos algorithm, that dramatically reduces the search space. We showed that the Kim and Cho algorithm, that enumerates the cartesian product of the matching substitutions operates in a larger search space, due to a much weaker reduction strategy. The other approach is based on a reduction of the matching candidates using context information for each literal. The context is given by occurrences of identical variables or chains of such occurrences. We presented two algorithms, that reflected this idea. We showed that there is a set of clauses, which is a superset of the determinate clauses, that can be tested for subsumption in polynomial time. The approaches are combined to a powerful algorithm: first the candidate sets are reduced using the graph context criterion, the remaining space is searched using the proposed clique algorithm.

Our empirical results, based on the finite element mesh design data set, show that both approaches strongly improve the performance, a combination of both approaches yields the best results, i.e. the least mean time and a small variance.

The determinate subsumption improves the performance in the negative case, although we could not observe an improvement in the positive case. In the negative case, there are very few problems that are extremely expensive.

The proposed combination of the context and the clique based subsumption algorithm can in turn be combined with the k-local match [GL85, KL94]. If C contains classes 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. Each class can be matched using the presented algorithm.

The efficiency of the θ-subsumption test is 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 w.r.t 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. We shortly completed the implementation of a first prototype of our learning system, that requires about 2 minutes to learn a fairly good hypothesis for the mesh design data set.

ACKNOWLEDGMENT

This work was partially supported by an Ernst-von-Siemens-Fellowship held by Tobias Scheffer. We wish to thank all our colleagues for their patience during the long time we performed our experiments on their workstations.

References

[BM92] B. Bolsak and S. Muggleton. The application of inductive logic programming to finite-element mesh design. In *Inductive Logic Programming*, London, 1992. Academic Press.

[CP90] R. Carraghan and P. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[DB93] L. DeRaedt and M. Bruynooghe. A theory of clausal discovery. In *Proc. Workshop on ILP*, 1993.

[DMR92] S. Dzeroski, S. Muggleton, and S. Russel. Pac-learnability of determinate logic programs. In *Proc. 5th ACM Workshop on Computational Learning Theory*, pages 128–135, 1992.

[Eis81] N. Eisinger. Subsumption and connection graphs. In *Proc. IJCAI*, 1981.

[FGL+91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating the maxclique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Sci.*, 1991.

[GHP96] L. Gibbons, D. Hearn, and P. Pardalos. A continuous based heuristic for the maximum clique problem. In *Clique, Graph Coloring and Satisfiability: Second DIMACS Implementation Challenge*, 1996.

[GL85] G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *J. ACM*, 32(2):280–295, 1985.

[Got87] G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24:109–111, 1987.

[GW96a] P. Geibel and F. Wysotzki. Learning relational concepts with decision trees. In *Proc. ICML*, 1996.

[GW96b] P. Geibel and F. Wysotzki. Relational learning with decision trees. In *Proc. ECAI*, 1996.

[JT96] D. S. Johnson and M. A. Trick, editors. *Clique, Graph Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS series, 1996.

[KC92] B. M. Kim and J. W. Cho. A new subsumption method in the connection graph proof procedure. *Theoretical Computer Science*, 103:283–309, 1992.

[KL94] J.-U. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In *Proc. International Conference on Machine Learning*, 1994.

[KN86] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In *Proc. 8th International Conference on Automated Deduction*, 1986.

[Kow75] R. Kowalski. A proof procedure using connection graphs. *J. ACM*, 22(4):572–595, 1975.

[Lov78] D. W. Loveland. *Automated theorem proving: A logical basis.* Elsevier, North Holland, 1978.

[MF90] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. 1st Conf. on Algorithmic Learning Theory*, pages 368–381, 1990.

[Mug93] S. Muggleton. Inverting implication. *Artificial Intelligence Journal*, 1993.

[Plo70] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163, 1970.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[Sic76] Sharon Sickel. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers*, C-25(8):823–835, 1976.

[Soc88] R. Socher. A subsumption algorithm based on characteristic matrices. In *Proc. 9th Int. Conf. on Automated Deduction*, 1988.

[Tin76] G. Tinhofer. Zum algorithmischen Nachweis der Isomorphie von endlichen Graphen. In H. Noltemeier, editor, *Graphen, Algorithmen, Datenstrukturen. 2. Fachtagung über Graphentheoretische Konzepte der Informatik.* Carl Hanser Verlag, 1976.

[UW81] S. Unger and F. Wysotzki. *Lernfähige Klassifizierungssysteme*. Akademie Verlag Berlin, 1981.

[vdLNC93] P. van der Laag and S. Nienhuys-Cheng. Subsumption and refinement in model inference. In *Machine Learning: ECML*, 1993.

[Wei76] B. Weisfeiler. *On Construction and Identification of Graphs*. Number 558 in Lecture Notes in Mathematics. Springer Verlag, Berlin, 1976.

[WSK81] F. Wysotzki, J. Selbig, and W. Kolbe. Concept learning by structured examples – an algebraic approach. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 1981.