Goal Lift-up: A Technique for Combining Connection-Tableau-Based Proof Procedures

Dirk Fuchs*
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: dfuchs@informatik.uni-kl.de

Abstract

Provers based on the connection tableau calculus and iterative deepening proof procedures belong to the most powerful theorem provers. Nevertheless, connection-tableau-based provers which are currently in use suffer from the inflexibility of the search bounds they employ for iterative deepening. We propose to overcome this problem by combining different iterative deepening procedures. In order to combine them we introduce the technique goal lift-up. We examine its abilities regarding proof length and search reduction and develop some heuristics so as to make the technique applicable in practice. A case study conducted in TPTP domains with the state-of-the-art theorem prover SETHEO demonstrates the strength of our approach.

1 Introduction

Provers based on the connection tableau calculus belong to the most powerful theorem provers. Reasons for this are their goal-oriented search and the possibility to use strong search pruning techniques like regularity (see [LMG94]) or matings pruning (see [Let98]). Moreover, the organization of their search by iterative deepening with backtracking ([Kor85]) offers some advantages. Highly efficient implementations utilizing structure sharing and abstract machine techniques are realizable. In addition, the search scheme supports further search pruning techniques like anti-lemmata.

However, iterative deepening procedures for theorem proving have not only advantages. The main problem of connection-tableau-based provers which are currently in use is the inflexibility of the search bounds that define the segments for iterative deepening. Depth-oriented search bounds suffer from the large increase of the initial proof segment when the admissible depth of tableaux is increased. As a consequence, only tableaux

*The author was supported by the *Deutsche Forschungsgemeinschaft (DFG)*.

which do not contain long branches can quickly be inferred. Provers based on inference-oriented bounds are weak in practice since these bounds only make very weak structural restrictions on the inferable tableaux (see [Har96]).

An approach to overcome the inflexibility of certain bounds is the *combination of different bounds*. So far, combination of inference- and depth-oriented bounds is achieved by developing bounds which contain inference- and depth-oriented elements (e.g., the weighted-depth bound [MIL+97]). Since such bounds are mainly depth-oriented due to performance reasons, the general problem cannot be removed completely.

In our approach, combination of bounds is indirectly achieved by *combining provers employing different bounds*. By combining different iterative deepening procedures different search bounds can support each other and overcome their weaknesses. In detail, our method is to *record subdeductions* which are performed employing a certain bound *in the form of clauses* (*compression of deductions*). If these clauses represent subdeductions which can hardly be performed when using other bounds, provers based on these bounds can utilize the clauses to reorganize their search and find proofs in smaller proof segments.

There exist (at least) two possible instantiations of our approach. The first one is the well-known technique of lemma exchange (see [Sch94, Fuc98e, Fuc98d]) which falls back on intermediate solutions of subgoals. Since [Fuc98d] and [Fuc98c] propose very efficient techniques for handling lemmas we are not going to deal with this technique here. Instead, we introduce the second possible instantiation *goal lift-up*. This technique is based on recorded transformations of original goal clauses into subgoal clauses.

Our combination approach can be realized with the help of *cooperative theorem provers*. Many provers work in parallel, generate lemmas or subgoal clauses, and exchange and use them. That is, our methods can be included in general frameworks for cooperative theorem proving, as e.g. the TECHS approach ([FD97, Fuc98b, DF98]).

The report is organized as follows. We start by explaining automated theorem proving with the connection tableau calculus in section 2. After that, in section 3 we point out in detail the problems of iterative deepening bounds that are currently in use. Furthermore, we introduce our approach for combining iterative deepening procedures. Section 4 addresses our combination approach goal lift-up. We define subgoal clauses and discuss in detail their capability of reducing proof lengths and proof searches. In addition, we introduce techniques for *filtering relevant subgoal clauses* which are unavoidable in order to obtain an efficient realization of our approach. In section 5, we proceed by describing a case study conducted with a cooperative system including the state-of-the-art prover SETHEO ([LSBB92]). It reveals the high potential of our methods. Finally, in section 6 we close with some concluding remarks.

2 Automated Theorem Proving with Connection Tableau Calculi

A typical top-down calculus is model elimination, essentially a restricted version of the *connection tableau calculus* (*CTC*) ([LMG94]). This calculus works on *connected* *tableaux* or *connection tableaux* for a clause[1] set $\mathcal{C}$. A tableau T for $\mathcal{C}$ is a tree whose non-root nodes are labeled with literals and that fulfills the condition: If the immediate successor nodes $v_1, \ldots, v_n$ of a node v of T are labeled with literals $l_1, \ldots, l_n$, then the clause $l_1 \vee \ldots \vee l_n$ (*tableau clause*) is an instance of a clause in $\mathcal{C}$. A tableau is called *connected* if each inner node v (non-leaf node) which is labeled with a literal l has a leaf node v' among its immediate successor nodes that is labeled with a literal l' complementary to l. A subtree of a connection tableau T whose nodes has the same labels as the respective nodes in T is called a *subtableau* of T.

The inference rules are *start*, *extension*, and *reduction*. For explaining the inference rules we need at first the notion of *expansion* ([Fit96]). An expansion step means selecting a clause from $\mathcal{C}$ and attaching the literals of a variant of this clause to a leaf node of an *open* branch, i.e. a branch that does not contain two complementary literals. The start rule allows for a standard tableau expansion that can only be applied to a trivial tableau, i.e. one consisting of only one node. We call the clause selected for the start expansion of a tableau T the *start clause* of T. Note that often also a restricted (but nevertheless complete) version of the calculus (that is called CTC_{neg}) is in use that only allows to perform the start expansion with negative clauses. Tableau reduction closes a branch by unifying a literal s at the leaf of an open tableau branch with the complement of a literal r (denoted by $\sim r$) on the same branch, and applying the substitution to the whole tableau. Extension is a combination of expansion and reduction. It is performed by selecting a literal s at the leaf of an open tableau branch, applying an expansion step to s, and immediately performing a reduction step with s and one of its newly created successors. Note that in the area of Horn clauses it is sufficient to employ start and extension, i.e. an explicit reduction inference is unnecessary. Thus, we assume that we use versions of CTC or CTC_{neg} that do not employ reduction in the area of Horn clauses.

A literal s at the leaf node of an open branch is called a *subgoal*. The subtree of a tableau T which contains all subgoal nodes of T (including the labels) and all ancestor nodes of these nodes in T (with labels) is called the *subgoal tree* of T. A literal s is *closed* or *solved* if it becomes the head literal of a closed subtableau after performing some inferences. A (sub-)tableau is *closed* if all branches are closed. The substitution stemming from the inferences needed to solve a subgoal s is called the *solution* of s.

CTC or CTC_{neg} do not have specific inference rules for handling equality. Instead, when dealing with equality the axiomatization must be extended by certain axioms, namely the reflexivity, symmetry, transitivity, and substitution axioms of equality.

We introduce some specific properties of tableaux. The *depth* $d(T)$ of a tableau T is the maximal depth of a node of T ignoring leaf nodes. The depth of the root node is 0, the depth of its immediate successor node in the tableau tree is 1, and so on. The *length* $l(T)$ of a tableau T is defined as the minimal number of inferences needed to infer a tableau T' from a trivial tableau which is equal to T modulo renaming variables. If T cannot be inferred starting from the trivial tableau $l(T)$ is undefined. The length of a tableau branch is defined as the number of literals lying on it. In order to simplify the notions in the following sections it is convenient to define the following if the empty

[1] A clause is a multiset of literals.

clause $\Box$ is an element of $\mathcal{C}$. Regardless of the use of CTC or CTC_{neg} we allow $\Box$ to be used for the start expansion. We consider the resulting tableau $T_\Box$ to be closed and define $d(T_\Box) = 0$ and $l(T_\Box) = 1$.

Important is the notion of a tableau derivation and a search tree: We say $T \vdash T'$ if (and only if) tableau T' can be derived from T by applying a start rule if T is the trivial tableau, or by an extension/reduction rule to a subgoal in T. Note that the calculus in use (CTC or CTC_{neg}) determines the relation $\vdash$. In order to show the unsatisfiability of a set $\mathcal{C}$, all derivations starting from the trivial tableau have to be enumerated until a closed tableau appears. Henceforth, we identify the notions 'closed (sub-)tableau' and '(sub-)proof'. The search space is given by a tableau search tree $\mathcal{T}^{\mathcal{C}}$ (if CTC is used) or $\mathcal{T}^{\mathcal{C}}_{neg}$ (if CTC_{neg} is used) defined as follows: A *search tree* $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}^{\mathcal{C}}_{neg}$) defined by a clause set $\mathcal{C}$ is given by a tree, whose root is labeled with the trivial tableau. Every inner node in $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}^{\mathcal{C}}_{neg}$) labeled with tableau T has as immediate successors the maximal set of nodes $\{v_1, \dots, v_n\}$, where v_i is labeled with T_i and $T \vdash T_i$, $1 \leq i \leq n$.

Since, in comparison with resolution style calculi, not only the number of proof objects but also their size increases during the proof process, explicit tableaux enumeration procedures that construct all tableaux in $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}^{\mathcal{C}}_{neg}$) in a breadth-first manner are not sensible. Hence, implicit enumeration procedures are normally in use that apply *consecutively bounded iterative deepening search with backtracking* ([Kor85]). In this approach iteratively larger finite initial parts of the search tree $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}^{\mathcal{C}}_{neg}$) are explored with depth-first search. Normally, a finite segment is defined by a so-called *completeness bound* (which poses structural restrictions on the tableaux which are allowed in the current segment) and a fixed natural number, a so-called *resource* (see [Sti88]). The segment of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}^{\mathcal{C}}_{neg}$) defined by a bound B and a resource k is denoted by $\mathcal{T}^{B,k,\mathcal{C}}$ ($\mathcal{T}^{B,k,\mathcal{C}}_{neg}$). Iterative deepening with a bound B is performed by starting with a basic resource value $n \in \mathbb{N}$ and iteratively increasing n until a proof is found within the finite initial segment $\mathcal{T}^{B,n,\mathcal{C}}$ ($\mathcal{T}^{B,n,\mathcal{C}}_{neg}$).

A prominent example for a completeness bound is the *inference bound* (see [Sti88]). The segment $\mathcal{T}^{Inf,k,\mathcal{C}}$ ($\mathcal{T}^{Inf,k,\mathcal{C}}_{neg}$) defined by the inference bound and a resource k contains all tableaux which can be inferred from the trivial tableau within k inferences. The depth bound limits the maximal tableau depth which is allowed in a segment $\mathcal{T}^{Depth,k,\mathcal{C}}$ ($\mathcal{T}^{Depth,k,\mathcal{C}}_{neg}$) according to a resource k. In practice, the depth bound is—in contrast to the inference bound—quite successful (cf. [LMG94, Har96]). The weighted-depth bound was introduced in [MIL+97]. This bound describes a class of possible bounds that restrict the tableau depth as well as the number of inferences allowed to infer a specific tableau. When choosing suitable parameters the bound can simulate both the depth and the inference bound. The configuration used within the prover SETHEO (see [MIL+97]) is rather depth-oriented and has proved to be quite successful.

3 Basics of Combining Connection-Tableau-Based Provers

This section addresses basics of combining tableau-based provers. At first, we deal with typical problems which arise from the use of common sequential iterative deepening procedures. After that, we point out in which way combining tableau-based provers is appropriate for overcoming the mentioned problems. Specifically, we introduce two combination techniques which can be realized by cooperating provers: *Goal lift-up* and *exchange of bottom-up lemmas*.

3.1 Problems of proof search with iterative deepening

The main problem of iterative deepening procedures for automated theorem proving is the inflexibility of the search bounds in use.

The rather powerful depth-oriented bounds, as e.g. the depth bound, suffer from the large increase of the initial segment of the search tree $\mathcal{T}^{B,k,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B,k,\mathcal{C}}$) if k is increased. As a consequence, closed tableaux which contain long branches are often out of reach of a prover based on a depth-oriented bound. This is because huge search spaces have to be explored so as to find such tableaux.

Provers using inference-oriented bounds, as e.g. the inference bound, have usually no problems in enumerating closed tableaux with few long branches. But these bounds are not very successful in practice. This is due to the fact that—especially if equality is involved in a proof problem—the same solutions of subgoals can be enumerated by various different subproofs of the same length (see, e.g., [Har96]). Hence, structural restrictions on the proof trees as used by depth-oriented bounds are more successful since they exclude many of these subproofs from initial segments of the search tree $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$). In addition, tableaux that have a high branching rate and whose enumeration requires many inferences can normally not be inferred by provers using inference-oriented bounds.

So far, these problems of iterative deepening bounds were tackled by the development of bounds (e.g. the weighted-depth bound) which consider both the depth of a tableau and the number of inferences necessary to infer it. However, due to efficiency reasons these bounds are mainly depth-oriented. Thus, provers employing them have problems in enumerating tableaux with some long branches, too.

3.2 Combining iterative deepening procedures

A possible solution to the mentioned problems is the combination of theorem provers that employ different iterative deepening bounds. Assume that some theorem provers $\wp_1, \ldots, \wp_m$ are given that use the same calculus and execute a common iterative deepening algorithm for theorem proving, but have different iterative deepening bounds $B_1, \ldots, B_m$. Then, our approach is as follows. Assume that there exist certain subdeductions (represented by subtableaux) that can be (and are) performed by a prover $\wp_i$ using a rather small resource but cannot be performed by some provers $\wp_z$, $z \neq i$, within an initial segment $\mathcal{T}^{B_z,k,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B_z,k,\mathcal{C}}$) with a rather small resource k. Then, such subdeductions should be *represented by clauses*. These clauses should be used by provers $\wp_z$ to *replace whole subdeductions* by just *one inference step*. If this is possible, the resource necessary to enumerate certain proofs (and hence also the runtime) may dramatically be decreased.

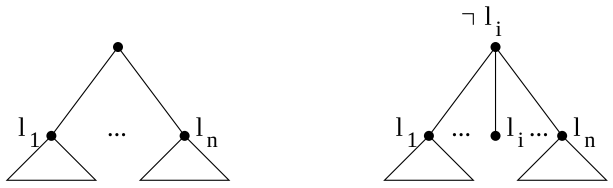


Figure 1: Different possibilities of clause use

We distinguish between two different ways of using clauses to replace subdeductions. On the one hand, a clause $C = l_1 \vee \ldots \vee l_n$ might represent a transformation of an original start clause O into new subgoals. As described in section 4 in more detail, such a clause could be used as a new start clause (see also the left hand side of Figure 1). The solution of the subgoals l_i within a segment $\mathcal{T}^{B,k,\mathcal{C}\cup\{C\}}$ ($\mathcal{T}_{neg}^{B,k,\mathcal{C}\cup\{C\}}$) with a rather small resource k is desirable: If the size of the minimal initial segment of $\mathcal{T}^{\mathcal{C}\cup\{C\}}$ ($\mathcal{T}_{neg}^{\mathcal{C}\cup\{C\}}$) which contains a proof when starting with C is smaller than the size of the minimal segment of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$) containing a proof when starting with O, high speed-ups are possible.

On the other hand, a clause $C = l_1 \vee \ldots \vee l_n$ can result from a subdeduction which solved a subgoal $\sim l_i$ during the proof run (see, e.g., [AS92]). Such a clause, a so-called *lemma*, could be used by other provers $\wp_z$ in order to solve the subgoal $\sim l_i$ (or subgoals unifiable with $\sim l_i$) during the proof run. If the solutions of these subgoals with bounds B_z normally require a high resource the use of C can significantly decrease the needed resource and hence the size of the minimal segment of the search tree which contains a proof (see also the right hand side of Figure 1). Naturally, speed-ups can only be expected if the solution of the remaining subgoals l_j, $j \neq i$, within a segment $\mathcal{T}^{B,k,\mathcal{C}\cup\{C\}}$ ($\mathcal{T}_{neg}^{B,k,\mathcal{C}\cup\{C\}}$) with a small resource k is possible.

All in all we can say that the first technique—which is in the following called *goal lift-up*—realizes some kind of *top-down lemma mechanism* since transformations of original proof goals (start clauses) are used. In contrast, *bottom-up lemmas* that stem from the intermediate solution of subgoals introduce bottom-up aspects into the proof search. There exist very efficient methods for generating and using such bottom-up lemmas. Hence, we are not going to deal with this topic here. The technique goal lift-up, however, is novel and will be studied in the following.

4 Goal Lift-Up for Combining Analytic Provers

We introduce the meta-rule *goal lift-up* which is well-suited for combining different analytic provers. We start by explaining the principles of goal lift-up. After that, theoretical and practical issues of employing the meta-rule are examined.

4.1 Principles of goal lift-up

The general idea of the meta-rule goal lift-up is to let each prover $\wp_i$ (employing bound B_i) perform some transformations of original goal clauses (which are the start clauses of the inferable tableaux) into new subgoals (forming *subgoal clauses*). Then, we lift these subgoal clauses up, i.e. other provers $\wp_z$ (with bounds B_z) should use them as new start clauses for tableaux enumeration. Note that only negative clauses are lifted up if CTC_{neg} is employed.

In order to define goal lift-up we need at first an exact definition of subgoal clauses.

Definition 4.1 subgoal clause, subgoal clause set

1. Let $\mathcal{C}$ be a set of clauses, let T be a tableau for $\mathcal{C}$. A *subgoal clause* S_T regarding T is the clause $S_T = l_1 \vee \ldots \vee l_n$, where the literals l_i are the subgoals of the tableau T.

2. Let B be a bound, n be a resource, and $\mathcal{C}$ be a set of clauses. If CTC is used, the *subgoal clause set* $\mathcal{S}^{B,n,\mathcal{C}}$ w.r.t. B, n, and $\mathcal{C}$, is defined by $\mathcal{S}^{B,n,\mathcal{C}} = (\cup S_T) \setminus \mathcal{C}$, T is a tableau which is a label of a node in $\mathcal{T}^{B,n,\mathcal{C}}$. If CTC_{neg} is in use, the *subgoal clause set* $\mathcal{S}^{B,n,\mathcal{C}}_{neg}$ is the set of subgoal clauses $\mathcal{S}^{B,n,\mathcal{C}}_{neg} = \{S_T : S_T \in \mathcal{S}^{B,n,\mathcal{C}}$, the start expansion of T is performed with a negative clause$\}$. $\diamond$

Note that subgoal clauses $\mathcal{S}^{B,n,\mathcal{C}}$ ($\mathcal{S}^{B,n,\mathcal{C}}_{neg}$) are valid clauses, i.e. logic consequences of $\mathcal{C}$ ([LMG94]). Further, $\Box$ is the subgoal clause of a closed tableau. The following example illustrates the above definition.

Example 4.1 Let $\mathcal{C} = \{\neg g, \neg p_1 \vee \ldots \vee \neg p_n \vee g, \neg q_1 \vee \ldots \vee \neg q_m \vee g\}$. Then, $\neg p_1 \vee \ldots \vee \neg p_n$ is the subgoal clause S_T belonging to the tableau obtained when extending the goal $\neg g$ with the clause $\neg p_1 \vee \ldots \vee \neg p_n \vee g$. If we employ $B =$ inference bound and resource $k = 2$, then $\mathcal{S}^{B,k,\mathcal{C}} = \mathcal{S}^{B,k,\mathcal{C}}_{neg} = \{\neg p_1 \vee \ldots \vee \neg p_n, \neg q_1 \vee \ldots \vee \neg q_m\}$. $\diamond$

As one can see, a subgoal clause S_T represents a transformation of an original goal clause (which is the start clause of the tableau T) into new subgoals realized by the deduction which led to the tableau T. A subgoal clause set represents all goal transformations which are possible in an initial part of the search tree.

Definition 4.2 goal lift-up

Let $\mathcal{C}$ be a set of clauses. Let B be an iterative deepening bound. For calculus CTC the rule *goal lift-up* is defined on sets of clauses by

$$\frac{\mathcal{M}}{\mathcal{M} \cup \{S_T\}}; \quad \exists k : S_T \in \mathcal{S}^{B,k,\mathcal{C}}$$

For calculus CTC_{neg} we define goal lift-up by

$$\frac{\mathcal{M}}{\mathcal{M} \cup \{S_T\}}; \quad S_T \text{ is negative}, \exists k : S_T \in \mathcal{S}^{B,k,\mathcal{C}}_{neg}$$

$\diamond$

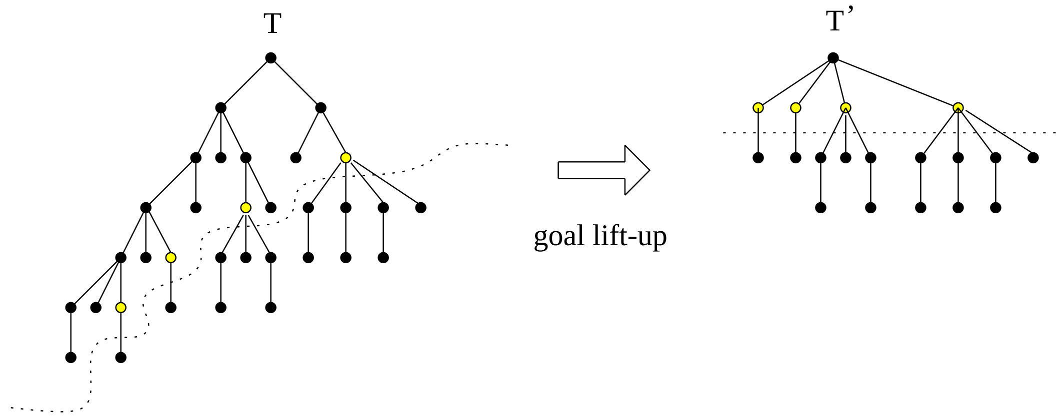


Figure 2: Resource reduction by goal lift-up

Note that the rule goal lift-up is a meta-rule for transforming input clause sets. As already mentioned, the intended use is as follows. We employ provers $\wp_1, \ldots, \wp_n$ which use the same calculus and the same algorithm for iterative deepening but different bounds $B_1, \ldots, B_n$. First, each prover $\wp_i$ $(1 \leq i \leq n)$ generates subgoal clauses from $\mathcal{S}^{B_i,k_i,\mathcal{C}}$ $(\mathcal{S}_{neg}^{B_i,k_i,\mathcal{C}})$ with a fixed resource k_i. After that, with the help of goal lift-up, each prover $\wp_i$ extends its input set with the generated subgoal clauses of *each other prover* $\wp_z$, $z \neq i$. It is also possible that a prover applies lift-up to its own generated clauses (see section 5). Finally, each prover tries to find a refutation of its new input set with its calculus. Henceforth, we consider calculi CTC and CTC_{neg} that employ subgoal clauses *only* for the start expansion. That is, these clauses must not be used for extension steps during the proof run.

As the following example shows, the combination of iterative deepening bounds by goal lift-up can give rise to high resource reductions, i.e. to a reduction of the minimal resource needed to infer a closed tableau.

Example 4.2 Let $\mathcal{C}$ be a set of clauses. Let $\wp_1$ and $\wp_2$ be provers based on the inference and depth bound, respectively. Let T be a closed tableau for $\mathcal{C}$ with minimal depth and minimal length. The structure of T is displayed on the left-hand side of Figure 2. As one can see, the depth of T is 5 and the length is 17, i.e. rather high inference and depth resources are necessary in order to infer T. As a consequence, usually neither $\wp_1$ nor $\wp_2$ are able to infer T in a short proof run.

However, with resource $i = 8$ prover $\wp_1$ can transform the original start clause into subgoals as displayed on the left-hand side of Figure 2: literals above the dotted line are inferred by $\wp_1$ and open subgoals are represented by grey circles. Note that $\wp_1$ is able to infer literals from long branches of T since it uses an inference-oriented bound. The right-hand side of Figure 2 shows the proof tableau T' after the lift-up of the subgoal clause containing the grey subgoals and its use as a new start clause. Now, a proof in depth 2 is possible, i.e. a closed tableau may quickly be inferred by prover $\wp_2$.

Note that also $\wp_2$ is able to generate subgoal clauses that are useful for $\wp_1$. E.g., with resource $d = 3$ prover $\wp_2$ can solve all subgoals on the right-hand side of tableau T such that the remaining subgoals can be solved with 6 inferences. Then, prover $\wp_1$ might be able to solve these subgoals and find a closed tableau with a small inference resource.$\diamond$

input: $\mathcal{C}$, B , $\Delta_{max} > \Delta_{min} > 0$

$k := 1; \mathcal{S} := \emptyset;$
$\mathcal{H} := \mathcal{S}^{B,k,\mathcal{C}}_{(neg)};$
`while` $|\mathcal{H}| < \Delta_{max}$ `do`
{
    $k := k + 1; \mathcal{S} := \mathcal{H};$
    $\mathcal{H} := \mathcal{S}^{B,k,\mathcal{C}}_{(neg)};$
}
`if` $|\mathcal{S}| < \Delta_{min}$ `then`
{
    $\mathcal{S} := \mathcal{H};$
}
`identify all variants of clauses in` $\mathcal{S}$
$\mathcal{S} := \{C \in \mathcal{S} : (\nexists D \in \mathcal{S} : D \neq C, D$ `subsumes` $C)\}$

output: $\mathcal{S}$

Figure 3: Algorithm for subgoal clause generation

It is to be emphasized that the lift-up meta-rule *should not be applied to all subgoal clauses* that represent a transformation of an original start clause within a given resource. As we will explain shortly this can entail that a resource reduction occurs but that the minimal proof segment (with the new smaller resource) is not smaller than the old minimal proof segment. Therefore, the lift-up rule should only be applied to few subgoal clauses *which are selected* from a *pool of generated subgoal clauses*. Then, goal lift-up can be interpreted as increasing flexibility of conventional bounds by allowing for more resources when solving certain selected subgoals. Increasing flexibility of bounds with goal lift-up is viable, in contrast to the development of a bound that gives—regarding certain criteria—additional resources to some subgoals. This is because such a new bound would require *absolute criteria* (which can hardly be developed) in order to judge whether certain subgoals should obtain additional resources. Goal lift-up, however, only requires *relative criteria* in order to identify certain subgoals (and hence subgoal clauses) that appear to be more promising than others. As Section 4.3 shows for this purpose powerful criteria can be developed. In addition, goal lift-up allows for a specific form of tableau subsumption as described shortly.

4.2 Generation of subgoal clauses

Henceforth, we want to deal with the generation of a pool of subgoal clauses from which some clauses can be selected (see section 4.3). When generating subgoal clauses it is on the one hand important that a resource being sufficiently large is used. A higher resource increases the chance to obtain a higher resource reduction. On the other hand, we must take care in the fact that the subgoal clause sets can become huge if they are generated regarding high resources. Then, potentially occurring speed-ups through subgoal clauses can be outweighed by the large costs to generate and to select them. Hence, our approach is to iteratively identify the maximal resource value k such that the size of the set $\mathcal{S}^{B,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{B,k,\mathcal{C}}$) is in a certain range.

The algorithm from Figure 3 is one possible approach for generating subgoal clauses. For a clause set $\mathcal{C}$, a bound B, and parameters $\Delta_{min}, \Delta_{max} \in \mathbb{N}$ ($0 < \Delta_{min} < \Delta_{max}$), it computes a subset of the subgoal clauses $\mathcal{S}^{B,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{B,k,\mathcal{C}}$) such that k is maximal with $|\mathcal{S}^{B,k,\mathcal{C}}| < \Delta_{max} \vee |\mathcal{S}^{B,k-1,\mathcal{C}}| < \Delta_{min}$ ($|\mathcal{S}_{neg}^{B,k,\mathcal{C}}| < \Delta_{max} \vee |\mathcal{S}_{neg}^{B,k-1,\mathcal{C}}| < \Delta_{min}$). Hence, the algorithm tries to limit the number of subgoal clauses by Δ_{max}. However, if this would result in a set with a size smaller than Δ_{min} we allow for the generation of more than Δ_{max} clauses. Note that after the generation of subgoal clauses subsumed clauses are deleted. This is because if a clause C subsumes a clause D then any refutation proof starting from D can be simulated when starting with C. Hence, this realizes a kind of tableau subsumption.

It is interesting to examine the potential of the set of generated subgoal clauses to reduce proof lengths and searches. More exactly, we are interested in two questions. Assume that $\mathcal{C}$ is an inconsistent set of clauses, that a theorem prover uses bound B_F for the final proof run, and that subgoal clauses were generated by another prover with bound B_P. Firstly, we should examine whether or not shorter proofs (w.r.t. the number of inferences) for the inconsistency of $\mathcal{C} \cup \mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{C} \cup \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) exist than for the inconsistency of $\mathcal{C}$. Since the generation of subgoal clauses requires some inferences it is interesting whether these inferences are saved when refuting the input set augmented with subgoal clauses. Secondly, it is interesting to compare the size of $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ and $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}^{B_P,k,\mathcal{C}}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$ and $\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}^{B_P,k,\mathcal{C}}}$) which are minimal segments of $\mathcal{T}^{\mathcal{C}}$ and $\mathcal{T}^{\mathcal{C}\cup\mathcal{S}^{B_P,k,\mathcal{C}}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$ and $\mathcal{T}_{neg}^{\mathcal{C}\cup\mathcal{S}_{neg}^{B_P,k,\mathcal{C}}}$) that contain a proof for the inconsistency of $\mathcal{C}$ and $\mathcal{C} \cup \mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{C}$ and $\mathcal{C} \cup \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$), respectively. If the size of the minimal proof segment when using subgoal clauses is smaller than the size of the minimal proof segment when not using subgoal clauses it is very probable that proofs can be found faster.

4.2.1 Reduction of proof lengths

For our study concerning proof lengths we only consider the inference and the depth bound. This is sensible because these are *the* typical inference- and depth-oriented bounds. We separately examine the areas of Horn and non-Horn problems and start with Horn logic.

Theorem 4.1 *Let $\mathcal{C}$ be an inconsistent set of Horn clauses, let $\Box \notin \mathcal{C}$. Let $B_P \in \{Depth, Inf\}$. If $B_P = Depth$, let $k \geq 1$, otherwise $k \geq 2$. The set $\mathcal{S}$ ($\mathcal{S}_{neg}$) results from the set $\mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) by deleting subsumed clauses. Further, let T_1 be a closed tableau for $\mathcal{C}$ with minimal length regarding CTC (CTC_{neg}), let T_2 be a closed tableau for $\mathcal{C} \cup \mathcal{S}$ ($\mathcal{C} \cup \mathcal{S}_{neg}$) with minimal length regarding CTC (CTC_{neg}). Then, it holds: $l(T_1) > l(T_2)$.*
Specifically, if $B_P = Inf$ then $l(T_2) = \max(\{1, l(T_1) - k + 1\})$.

Proof: Let $B_P = Depth$, let $T_{\min}$ be a closed tableau for $\mathcal{C}$ with minimal length regarding CTC (CTC_{neg}). Then, by using a subgoal clause from $\mathcal{S}$ ($\mathcal{S}_{neg}$) as start clause the depth of the tableau $T_{\min}$ can be reduced by $\min(\{k, d(T_{\min})\})$. Since $\min(\{k, d(T_{\min})\}) \geq 1$ the number of inferences needed to infer a tableau of minimal length is reduced.
If $B_P = Inf$, when using subgoal clauses as start clauses obviously the length of each tableau T can be reduced by $\min(\{k-1, l(T)-1\})$, hence also the length of a closed tableau for $\mathcal{C}$ of minimal length. □

Hence, in the area of Horn logic minimal proof lengths can always be reduced. It is even possible to reduce the length of each closed tableau. The reduction of the proof length when using the depth bound for the generation of subgoal clauses depends on the branching rates of the tableaux T that lead to subgoal clauses S_T. Note that generating subgoal clauses with the depth bound and resource k can decrease the proof length by an amount much greater than k.

The proof of Theorem 4.1 shows that subgoal clauses $S_T \in \mathcal{S}^{B_P,k,\mathcal{C}}$ ($S_T \in \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) where $\forall T' : T \vdash T' \Rightarrow S_{T'} \in \mathcal{S}^{B_P,k,\mathcal{C}}$ ($\forall T' : T \vdash T' \Rightarrow S_{T'} \in \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) are unnecessary for proof length reductions. This is because starting with one clause $S_{T'}$ leads to shorter proofs than starting with S_T. Hence, the set of subgoal clauses can further be reduced.

In the area of non-Horn problems reduction steps might be used in a proof of minimal length. When generating subgoal clauses we do not consider the path context of the subgoals that form a subgoal clause, i.e. the literals on the branches to each subgoal. If reduction steps from outside are needed in the subproofs of the subgoals it might be that proof lengths cannot be reduced when starting the refutation proof with a subgoal clause.

Theorem 4.2 *Let $B_P \in \{Depth, Inf\}$. If $B_P = Depth$, let $k = 1$, otherwise let $k = 2$. Then an inconsistent clause set $\mathcal{C}$ exists with following property: no closed tableau for $\mathcal{C} \cup \mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{C} \cup \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) with minimal length regarding CTC (CTC_{neg}) is shorter than a minimal closed tableau for $\mathcal{C}$ regarding CTC (CTC_{neg}).*

Proof: Regardless of the use of CTC or CTC_{neg}, let

$$\begin{aligned} \mathcal{C} &= \{p_1(x) \vee p_2(y), \neg p_1(x) \vee p_2(y), \\ &\quad\; p_1(x) \vee \neg p_2(y), \neg p_1(x) \vee \neg p_2(y)\} \end{aligned}$$

It is obvious that a proof of minimal length for $\mathcal{C}$ requires 7 inferences. For $B_P = Inf$ (and $k = 2$) as well as for $B_P = Depth$ (and $k = 1$) we obtain:

$$\begin{aligned} \mathcal{S}_{neg}^{B_P,k,\mathcal{C}} &= \{\neg p_1(x) \vee \neg p_1(y), \neg p_1(x) \vee p_1(y), \\ &\quad\; \neg p_2(x) \vee p_2(y), \neg p_2(x) \vee \neg p_2(y)\} \end{aligned}$$

Further, $\mathcal{S}^{B_P,k,\mathcal{C}} = \mathcal{S}_{neg}^{B_P,k,\mathcal{C}} \cup \{p_1(x) \vee p_1(y), p_2(x) \vee p_2(y)\}$. Again, it is easy to see that when starting with subgoal clauses—regardless of the use of CTC or CTC_{neg}—a proof of minimal length requires 7 inferences. Hence, the minimal proof length cannot be reduced. □

4.2.2 Reduction of proof searches

Henceforth, we are going to examine whether or not subgoal clauses give rise to a resource reduction. As already discussed, we are interested in the following question. Assume that $\mathcal{C}$ is an inconsistent set of clauses, $\Box \notin \mathcal{C}$, that B_P is the bound that was used for the generation of subgoal clauses (with resource k), and that B_F is the bound for the final proof run. The set of generated subgoal clauses $\mathcal{S}$ ($\mathcal{S}_{neg}$) results from the set $\mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) by identifying variants and eliminating subsumed clauses. Let $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$) be the minimal segment of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$) that contains a closed tableau for $\mathcal{C}$ and $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$) be the minimal segment of $\mathcal{T}^{\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{\mathcal{C}\cup\mathcal{S}_{neg}}$) that contains a closed tableau for $\mathcal{C}\cup\mathcal{S}$ ($\mathcal{C}\cup\mathcal{S}_{neg}$). Then, we want to compare the size of these minimal segments.

If $k_2 = k_1$, the proof search for a refutation proof for $\mathcal{C}\cup\mathcal{S}$ ($\mathcal{C}\cup\mathcal{S}_{neg}$) is usually much more difficult than the search for a refutation proof for $\mathcal{C}$. This is because the branching rate of the search tree $\mathcal{T}^{\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{\mathcal{C}\cup\mathcal{S}_{neg}}$) is much larger than the branching rate of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$). Hence, improvements of the search can mainly be expected if $k_2 < k_1$. However, also in this case it is unclear whether the size of the minimal proof segment is really decreased and a proof may be found faster.

Again, we examine Horn and non-Horn problems and start with Horn problems.

Horn problems: Let $B_P = \mathit{Depth}$. As we have seen in the proof of Theorem 4.1 the depth of all closed tableaux T for $\mathcal{C}$ can be reduced by $r = \min(\{k, d(T)\})$ when using a subgoal clause as start clause. If $B_F = \mathit{Depth}$, we hence have a resource reduction by the amount of r, i.e. $k_2 = k_1 - r$. If we compare the structure of the segments $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ and $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$ and $\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$) we obtain: There are no subgoals in tableaux from $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$) that do not occur in tableaux from $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$). Further, no additional solutions of subgoals can be obtained. But it might be that identical subgoals have to be solved more often when using subgoal clauses because they occur when starting with a clause from $\mathcal{C}$ and when starting with a subgoal clause. Hence, the use of subgoal clauses may increase the redundancy. It is to be emphasized, however, that due to the subsumption test from our algorithm of Figure 3 certain solutions of subgoals that can be found in $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$) cannot be found in $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$). In the case that $B_F = \mathit{Inf}$ we obtain a resource reduction by an amount of n. The exact value of n depends—as already mentioned—on the structure of the tableaux T that lead to subgoal clauses S_T. In general, we obtain similar results about the structure of the minimal proof segments as in the case $B_F = \mathit{Depth}$. There can be multiple occurrences of the same subgoals when using subgoal clauses, but the subsumption test can save subsumed solutions of subgoals. But we should note that an additional problem occurs. It is possible to obtain additional solutions of subgoals if they can profit more from the use of subgoal clauses than a proof of minimal length can. Since the depth bound is used for the preprocessing the lengths of different proofs need not be reduced by the same value. Hence, it might be that solutions of subgoals can be found in the minimal segment when using subgoal clauses but not in the minimal segment when not using subgoal clauses.

If $B_P = \mathit{Inf}$, $B_F = \mathit{Inf}$, and T is a closed tableau for $\mathcal{C}$ of minimal length, always a resource reduction by $r = \min(\{k-1, l(T)-1\})$ takes place. The structure of the minimal proof segments is as in the case $B_P = B_F = Depth$. If we use $B_F = Depth$ the guaranteed proof length reduction from Theorem 4.1 need not give rise to a proof depth (= resource) reduction as following example shows.

Example 4.3 Let $\mathcal{C} = \{\neg p_1 \vee \neg p_2, p_1, p_2\}$. Let $k = 2$, $B_P = Inf$, and $B_F = Depth$. Then, each proof for $\mathcal{C}$ with CTC or CTC_{neg} has the depth 1. Further, $\mathcal{S}^{B_P,k,\mathcal{C}} = \mathcal{S}_{neg}^{B_P,k,\mathcal{C}} = \{\neg p_1, \neg p_2\}$. Thus, each proof for $\mathcal{C} \cup \mathcal{S}^{B_P,k,\mathcal{C}}$ ($\mathcal{C} \cup \mathcal{S}_{neg}^{B_P,k,\mathcal{C}}$) has depth 1, too. Note that the proof length is reduced when using subgoal clauses. $\diamond$

All in all we obtain that for Horn problems and $B_P = Depth$ always resource reductions take place. This usually does not entail that the minimal proof segment for refuting $\mathcal{C} \cup \mathcal{S}$ ($\mathcal{C} \cup \mathcal{S}_{neg}$) is smaller than the minimal proof segment for refuting $\mathcal{C}$. Depending on the number of subsumed solutions of subgoals that exist the minimal segment may be increased or decreased. We achieve the same result for $B_P = B_F = Inf$. For $B_P = Inf$ and $B_F = Depth$ no resource reduction need occur, i.e. the proof search when employing subgoal clauses may be more complicated than without subgoal clauses.

Thus, it is reasonable to *select relevant subgoal clauses* from $\mathcal{S}$ ($\mathcal{S}_{neg}$). If already a few selected clauses give rise to a resource reduction a significant decrease of the minimal proof segment and hence a significant speed-up of the search can occur. This is because many inferences needed to obtain solutions of subgoals cannot be performed in a smaller segment and can also not be simulated with the help of subgoal clauses. Note that the number of inferences needed to infer the subgoal clauses is usually much smaller than the number of inferences which are saved. Due to the exponential increase of the size of proof segments with increasing resource it is often more efficient to enumerate tableaux from two segments of the search tree defined by a small resource than to enumerate tableaux from a segment defined by a larger resource.

Non-Horn problems: When considering non-Horn problems and $B_P \in \{Inf, Depth\}$ we can observe that for $B_F = Inf$ no resource reductions need take place since no proof length reductions are guaranteed (see Theorem 4.2). At least each occurring proof length reduction reduces the resource. As following example shows, if $B_F = Depth$ there is also no resource reduction guaranteed. Even more, possibly occurring proof length reductions need not decrease the resource needed to infer a closed tableau.

Example 4.4

1. The case $B_P = Inf$, $B_F = Depth$ is exactly in analogy to Example 4.3.

2. $B_P = Depth$, $B_F = Depth$. Let $\mathcal{C} = \{p_1 \vee p_2, \neg p_1 \vee p_2, p_1 \vee \neg p_2, \neg p_1 \vee \neg p_2 \vee \neg p_3, p_3\}$. Each proof of minimal depth for $\mathcal{C}$ has depth 2. By using resource $k = 1$ for the generation of subgoal clauses we obtain by subsumption deletion from $\mathcal{S}_{neg}^{B_P,1,\mathcal{C}}$ and $\mathcal{S}^{B_P,1,\mathcal{C}}$ the sets $\mathcal{S}_{neg} = \{\neg p_1 \vee \neg p_2, \neg p_1 \vee \neg p_1, \neg p_1 \vee p_1, \neg p_2 \vee \neg p_2, \neg p_2 \vee p_2\}$ and $\mathcal{S} = \mathcal{S}_{neg} \cup \{p_1 \vee p_1, p_2 \vee p_2\}$. Thus, each proof of minimal depth for $\mathcal{C} \cup \mathcal{S}^{B_P,1,\mathcal{C}}$ ($\mathcal{C} \cup \mathcal{S}_{neg}^{B_P,1,\mathcal{C}}$) has depth 2, too. Note that the proof length is reduced when using subgoal clauses because subgoal $\neg p_3$ can be closed during the generation of subgoal clauses. $\diamond$

Hence, the situation for non-Horn problems is worse than the situation for Horn problems since resource reductions need not take place. However, if a resource reduction from k_1 to k_2 ($k_2 < k_1$) is possible when using subgoal clauses many inferences can be saved. Firstly, when using CTC_{neg} we can save inference chains possible in $\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$ but not in $\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$ where extension steps are applied to positive subgoals. These chains cannot be simulated with subgoal clauses. Secondly, certain inference chains possible in $\mathcal{T}^{B_F,k_1,\mathcal{C}}$ ($\mathcal{T}_{neg}^{B_F,k_1,\mathcal{C}}$) but not in $\mathcal{T}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}}$ ($\mathcal{T}_{neg}^{B_F,k_2,\mathcal{C}\cup\mathcal{S}_{neg}}$) which contain reduction steps cannot be simulated. As in the case of Horn problems very large proof search reductions can be expected if already few selected subgoal clauses allow for a resource reduction.

4.2.3 Dealing with non-Horn clauses

The main problem when dealing with non-Horn clauses that we have identified in section 4.2.1 is that during the generation of subgoal clauses the literals on the path to each subgoal are lost. Thus, they cannot be used for reduction entailing negative consequences on possible proof length reductions.

A simple solution to this problem is the following. During the generation of subgoal clauses so-called *context literals* are stored for each subgoal. Then, the calculi CTC and CTC_{neg} are modified in such a way that context literals can be used for reduction steps. As we will see, for calculus CTC these modifications allow for lifting the results from the Horn to the non-Horn case.

Generating and using subgoal clauses with context: Henceforth, we call $l_1|M_1 \vee \ldots \vee l_n|M_n$ where the l_i are literals and M_i are sets of literals a *clause with context*. First, we define subgoal clauses with context.

Definition 4.3 subgoal clause with context, subgoal clause set with context

1. Let T be a tableau. Let $l_1, \ldots, l_m$ be the subgoals of T. For each subgoal l_i, let $n_1^i, \ldots, n_{k_i}^i$ (marked with literals $p_1^i, \ldots, p_{k_i}^i$) be the inner nodes of the branch whose leaf node is marked with l_i. Then, the *subgoal clause with context* S_T is defined by
$$S_T = l_1|\{p_1^1, \ldots, p_{k_1}^1\} \vee \ldots \vee l_m|\{p_1^m, \ldots, p_{k_m}^m\}$$
2. Let B be a bound, n be a resource, and $\mathcal{C}$ be a set of clauses. If CTC is used, the *subgoal clause set with context* $\mathcal{S}^{B,n,\mathcal{C}}$ w.r.t. B, n, and $\mathcal{C}$, is defined by $\mathcal{S}^{B,n,\mathcal{C}} = (\cup S_T) \setminus \mathcal{C}$, T is a tableau which is a label of a node in $\mathcal{T}^{B,n,\mathcal{C}}$. If CTC_{neg} is in use, the *subgoal clause set with context* $\mathcal{S}_{neg}^{B,n,\mathcal{C}}$ is the set of subgoal clauses with context $\mathcal{S}_{neg}^{B,n,\mathcal{C}} = \{S_T : S_T \in \mathcal{S}^{B,n,\mathcal{C}}$, the start expansion of T is performed with a negative clause$\}$. $\diamond$

Following example illustrates the above definition.

Example 4.5 Let $\mathcal{C} = \{\neg g, \neg p_1 \vee \ldots \vee \neg p_n \vee g, \neg q_1 \vee \ldots \vee \neg q_m \vee g\}$. Then, $\neg p_1|\{\neg g\} \vee \ldots \vee \neg p_n|\{\neg g\}$ is the subgoal clause S_T belonging to the tableau obtained when extending the goal $\neg g$ with the clause $\neg p_1 \vee \ldots \vee \neg p_n \vee g$. If we employ $B =$ inference bound and resource $k = 2$, then $\mathcal{S}_{neg}^{B,k,\mathcal{C}} = \{\neg p_1|\{\neg g\} \vee \ldots \vee \neg p_n|\{\neg g\}, \neg q_1|\{\neg g\} \vee \ldots \vee \neg q_m|\{\neg g\}\}$. $\mathcal{S}^{B,k,\mathcal{C}} = \mathcal{S}_{neg}^{B,k,\mathcal{C}} \cup \{\neg p_1 \vee \ldots \vee \neg p_n, \neg q_1 \vee \ldots \vee \neg q_m\}$. $\diamond$

The meta rule goal lift-up with context can be defined in analogy to before. The only difference is that we lift up subgoal clauses with context instead of conventional subgoal clauses.

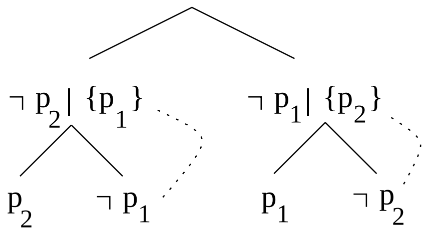
For the generation of subgoal clauses with context we can use the algorithm from Figure 3. However, we must slightly modify the definition of subsumption. If we have clauses with context $C = l_1|M_1 \vee \ldots \vee l_m|M_m$ and $D = s_1|N_1 \vee \ldots \vee s_n|N_n$ we say that C subsumes D if following holds. There is a substitution σ such that for each $l_i|M_i$ in C there is a $s_j|N_j$ in D with $l_i\sigma = s_j$ and $N_j \subseteq M_i\sigma$. This definition of subsumption guarantees that any refutation proof starting with D can also be found when starting with C. In the above example the clauses $\neg p_1 \vee \ldots \vee \neg p_n$ and $\neg q_1 \vee \ldots \vee \neg q_m$ are subsumed by others and can be deleted.

Up to now, we have made precise how to generate subgoal clauses that provide information on their path contexts. Since our conventional calculi CTC or CTC_{neg} cannot exploit this information we must slightly modify these calculi. Both calculi CTC^{con} and CTC_{neg}^{con} consist of the rules *start with context, extension, and reduction with context*. The only difference between CTC^{con} and CTC_{neg}^{con} is that the latter calculus only apply start with context to negative clauses. The inference rule start with context selects a clause $l_1 \vee \ldots \vee l_m$ or a clause with context $l_1|M_1 \vee \ldots \vee l_m|M_m$ from the initial clause set and attaches its literals l_i to the unmarked root. Further, each node n_i marked with l_i obtains as additional mark M_i, if a clause with context was selected, or $\emptyset$, if a "normal" clause was selected. Extension is defined as before, i.e. the context literals are ignored in this case. As before, we forbid subgoal clauses with context to take part in extension steps. Hence, only nodes that are immediate successor nodes of the unmarked root may additionally be marked with context literals. Reduction with context—applied to a leaf literal l—is defined in analogy to the normal reduction rule of CTC (CTC_{neg}). In addition, the context literals of a node on a branch to l can be used for reduction. Naturally, the substitution used for reduction is applied to the whole tableau (including the context literals).

Note that similar extensions of CTC and CTC_{neg} are used for realizing the folding-up ([LMG94]) and c−reduction ([Sho76]) rule. Hence, powerful provers like SETHEO ([MIL+97]) usually provide facilities for using clauses with context.

Before we analyze proof length and depth reductions in detail, we want to show with an example how these modified calculi work.

Example 4.6 Let $B_P = Depth$. Let $\mathcal{C} = \{p_1 \vee p_2, \neg p_1 \vee p_2, p_1 \vee \neg p_2, \neg p_1 \vee \neg p_2\}$. Then, a proof for $\mathcal{C}$ using CTC has depth 2. If we use CTC, B_P, and $k = 1$ for the generation of subgoal clauses, among others the subgoal clause $\neg p_2|\{p_1\} \vee \neg p_1|\{p_2\}$ is generated. With the help of this clause, the following proof of depth 1 can be found using calculus CTC^{con}.



Note that when using CTC or CTC_{neg} and subgoal clauses without context a proof depth reduction is impossible. $\diamond$

Proof length reduction: As the following theorem shows, at least for calculus CTC the results from the Horn case can be lifted to the non-Horn case when generating subgoal clauses with context and using our modified calculus for the refutation.

Theorem 4.3 *Let $\mathcal{C}$ be an inconsistent set of non-Horn clauses, let $\Box \notin \mathcal{C}$. Let $B_P \in \{Depth, Inf\}$. If $B_P = Depth$, let $k \geq 1$, otherwise $k \geq 2$ be the resource for generating subgoal clauses with context. The set $\mathcal{S}$ results from the set $\mathcal{S}^{B_P,k,\mathcal{C}}$ by deleting subsumed clauses. Further, let T_1 be a closed tableau for $\mathcal{C}$ with minimal length regarding CTC, let T_2 be a closed tableau for $\mathcal{C} \cup \mathcal{S}$ with minimal length regarding CTC^{con}. Then, it holds: $l(T_1) > l(T_2)$.*
Specifically, if $B_P = Inf$ then $l(T_2) = \max(\{1, l(T_1) - k + 1\})$.

Proof: In analogy to the proof of Theorem 4.1. $\Box$

Proof search reduction: For calculus CTC, the results regarding the proof search can be lifted from the Horn case, too. We obtain exactly the same results regarding the kinds of subgoals and the number of solutions of subgoals. It is to be emphasized that the selection of a few subgoal clauses that lead to proof length or resource reductions may drastically speed up the proof search.

4.3 Selection of subgoal clauses

If $\mathcal{C}$ is an inconsistent set of clauses, subgoal clauses $\mathcal{S}$ ($\mathcal{S}_{neg}$) should enable a prover to refute $\mathcal{C} \cup \mathcal{S}$ ($\mathcal{C} \cup \mathcal{S}_{neg}$) faster than $\mathcal{C}$. The best prospects in order to achieve this exist if there is a closed tableau for $\mathcal{C} \cup \mathcal{S}$ ($\mathcal{C} \cup \mathcal{S}_{neg}$) in a segment of the search tree $\mathcal{T}^{\mathcal{C} \cup \mathcal{S}}$ ($\mathcal{T}_{neg}^{\mathcal{C} \cup \mathcal{S}_{neg}}$) that is smaller than the minimal segment of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$) which contains a closed tableau. Obviously, this necessitates a resource reduction since the branching rate of $\mathcal{T}^{\mathcal{C} \cup \mathcal{S}}$ ($\mathcal{T}_{neg}^{\mathcal{C} \cup \mathcal{S}_{neg}}$) is greater than the branching rate of $\mathcal{T}^{\mathcal{C}}$ ($\mathcal{T}_{neg}^{\mathcal{C}}$). Moreover, our examination from the preceding section taught us that normally merely the use of a *small* set of subgoal clauses may entail that resource reductions really lead to a decrease of the minimal proof segment. Hence, techniques for a *relevancy-based selection* of subgoal clauses must be developed.

This section addresses principles of the relevancy-based selection of subgoal clauses from a given pool and introduces two approaches: *subgoal clause sensitive* selection and *subgoal tree sensitive* selection.

4.3.1 Principles

Assume that a pool $\mathcal{S}$ of subgoal clauses for an inconsistent clause set $\mathcal{C}$, $\Box \notin \mathcal{C}$, is given. Then, we are going to identify relevant subgoal clauses $\mathcal{R} \subset \mathcal{S}$ that allow for refuting $\mathcal{C} \cup \mathcal{R}$ in a smaller resource than $\mathcal{C}$. In order to do this, we fall back on a relevancy function ρ which associates subgoal clauses with natural numbers. The value which is associated with each clause is interpreted in such a way that high values indicate a high relevancy of the clause. The selection is performed by choosing subgoal clauses with the highest ratings w.r.t. ρ until a certain maximal number of subgoal clauses is chosen.

Before we define ρ we make at first precise on which structures ρ can operate. Let S_T be a subgoal clause which is extracted from the tableau T. Then, on the one hand ρ can operate on the syntactic structure of S_T only. If this is the case, we say that the selection is subgoal clause sensitive. On the other hand, ρ can use information on the derivation of S_T provided by the tableau T. More exactly, ρ might fall back on the subgoal tree $\hat{T}$ of T whose leaf nodes form the subgoal clause S_T. Then, we say that the selection is subgoal tree sensitive.

The best way is to combine both kinds of selection. Thus, we compute a subgoal clause sensitive part ρ_{sgc} and a subgoal tree sensitive part ρ_{sgt}, and ρ is defined by

$$\rho(S_T) = f_{sgc} \cdot \rho_{sgc}(S_T) + f_{sgt} \cdot \rho_{sgt}(S_T).$$

Note that we identify the subgoal clause S_T with its subgoal tree $\hat{T}$ in order to compute ρ_{sgt}. f_{sgc} and f_{sgt} are natural numbers that determine the influence of each selection technique.

4.3.2 Subgoal clause sensitive selection

The main characteristic of our approach is to locally judge each subgoal if it appears to be solvable with few inferences. In order to do this we use a *feature-based* approach. Some feature functions ℓ_i $(1 \leq i \leq k)$ are applied to each subgoal and map it to integer values. Each feature function judges—regarding a certain property of the subgoal—if it is possible to solve it with few inferences. A high feature value means that the subgoal seems to be easily solvable. Then, the relevancy of a subgoal clause is measured regarding the local judgment of its subgoals. Hence, we obtain

$$\rho_{sgc}(l_1 \vee \ldots \vee l_n) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_j \cdot \ell_j(l_i).$$

The a_j are natural numbers which weight the importance of certain properties. We developed two different feature functions ℓ_{gen} and ℓ_{sim} and weighted both with the value one.

ℓ_{gen} favors subgoals that are quite *general* and hence possibly solvable. Additionally, ℓ_{gen} considers whether a subgoal has a rather "flat" structure, i.e. not so many deep subterms. This may increase the number of possible solutions of the subgoal. We define

Figure 4: Subgoal tree sensitive selection: Part 1

ℓ_{gen} on negative literals $\sim l$ by $\ell_{gen}(\sim l) = \ell_{gen}(l)$ and on positive literals $l = P(t_1, \ldots, t_n)$ by $\ell_{gen}(l) = -\sum_{i=1}^{n} \ell_{gen}^{term}(t_i, 1)$. ℓ_{gen}^{term} is defined on pairs of terms and natural numbers by

$$\ell_{gen}^{term}(t, d) = \begin{cases} 1 + d & ; t \text{ is a variable} \\ 2 + d + \sum_{j=1}^{k} \ell_{gen}^{term}(t_j, d+1) & ; t = f(t_1, \ldots, t_k) \end{cases}$$

The feature function ℓ_{sim} measures a kind of similarity between unit clauses from the initial axioms and a subgoal. A high similarity between a unit clause C and a subgoal l should be a hint that C is possibly able to solve l or a descendant of l that can be derived with few inferences. We use a form of structural similarity by falling back on a variant *sim* of the function *occnest* as defined in [DF94]. Essentially, *sim* considers a unit C and a subgoal l to be similar if there are only small differences in the occurrences and nesting of function symbols in C and l. For a possible definition of such a function see [DF94]. If $\mathcal{C}$ is the set of input clauses, we obtain (when setting $\max(\emptyset) = 0$)

$$\ell_{sim}(l) = \max(\{sim(C, l) : C \in \mathcal{C}, |C| = 1\}).$$

4.3.3 Subgoal tree sensitive selection

Selecting subgoal clauses regarding their syntactic structure has the disadvantage that the deduction that led to subgoal clauses is ignored. The selection process can be improved if the subgoal trees whose leaf nodes form subgoal clauses are considered. E.g., we can use this information so as to estimate if a transformation into subgoal clauses is difficult w.r.t. a certain search bound. If this is the case, a prover employing this bound and the subgoal clauses can probably solve problems that were out of reach before. Note that we only perform a subgoal tree sensitive selection if the subgoal clauses are used by a prover based on a depth-oriented bound. Since inference-oriented bounds do not make restrictions on the structure of inferable subgoal trees such selection criteria are useless.

Before we try to judge the usefulness of subgoal clauses regarding the structure of their subgoal trees it is useful to take a look at typical structures of proofs found by depth-oriented bounds. Naturally, most closed tableaux found by depth-oriented bounds are quite symmetric, i.e. the length of different branches does not vary very much (type A). Closed tableaux that have few long branches and many short branches are usually out of reach of a depth-oriented bound (type B). Now, we want to find subgoal clauses (trees) that allow for finding tableaux from types A or B faster.

Figure 5: Subgoal tree sensitive selection: Part 2

We start with the search for tableaux from type A. Figure 4 shows two identical copies of a rather symmetric closed tableau. The subgoals above the dotted line on the left-hand side belong to subgoal clause S_{T_1}, the subgoals above the dotted line on the right-hand side to subgoal clause S_{T_2}. As we can see, the structure of the subgoal tree $\hat{T}_1$ is rather symmetric, i.e. all subgoals of S_{T_1} can be solved in a rather small depth. In contrast, the subgoal tree $\hat{T}_2$ associated with S_{T_2} is not symmetric. Hence, some subgoals can be solved in a small depth, others require a larger depth. All in all, S_{T_1} gives rise to a higher resource reduction and hence to faster proof runs when using a depth-oriented bound. This motivates the following definition of the quality of a subgoal clause $S_T = l_1 \vee \ldots \vee l_n$ represented by its subgoal tree $\hat{T}$.

$$\rho^A_{sgt}(l_1 \vee \ldots \vee l_n) = \min_{1 \le i \le n}(|\{b : b \neq l_i, b \text{ is a literal on the branch in } \hat{T} \text{ whose leaf is } l_i\}|).$$

When searching for proofs of type B we must identify subgoal clauses that enable a prover to close long branches. Figure 5 shows two identical copies of a closed tableau containing a long branch and many short ones. Usually, such a tableau is out of reach of a prover using a depth-oriented bound. The subgoals above the dotted line on the left-hand side of Figure 5 form the clause S_{T_1}, the subgoals above the dotted line on the right-hand side S_{T_2}. As we can see, by using S_{T_1} the depth of the long branch is reduced. Although the depth of other branches is not reduced very much a proof with a rather small resource is possible. When using S_{T_2} as start clause a closed tableau might be out of reach because the depth of the long branch is not reduced sufficiently. Note that it is important to find out if really a long branch can be shortened by a subgoal clause. Hence, there should be at least one branch in the subgoal tree of a subgoal clause whose length exceeds a parameter l_{min}. We get

$$\rho^B_{sgt}(l_1 \vee \ldots \vee l_n) = \max(\{0, \max_{1 \le i \le n}(|\{b : b \neq l_i, b \text{ is on the branch in } \hat{T} \text{ to } l_i\}|) - l_{min}).$$

Since we are interested in subgoal clauses that support the search for tableaux from both type A and B, we set

$$\rho_{sgt}(S_T) = \max(\rho^A_{sgt}(S_T), \rho^B_{sgt}(S_T)).$$

Note that subgoal clauses that support the search for tableaux from type A can mainly be found when generating subgoal clauses with a depth-oriented bound, whereas subgoal clauses that are useful for finding tableaux from type B can be found when generating subgoal clauses with inference-oriented bounds.

5 Experimental Study

Henceforth, we are going to examine the potential of our combination concept. First, we describe the experimental setting. After that, we proceed by presenting and discussing obtained results.

5.1 Experimental setting

Our experimental system is composed in the following way. The architecture is given by two theorem provers based on the connection tableau calculus, each running on its own processor. We employ variants of one basic prover that are obtained by using different iterative deepening bounds. The behavior of the system is characterized by three phases. At first, the provers tackle the proof problem in parallel for 5 seconds. Hence, trivial problems can be solved during this time. If the problem remains unsolved, in phase 2 we let the provers cooperate. We extract subgoal clauses from their unsuccessful proof runs (see below) and exchange the extracted clauses. After the augmentation of the original clause set with subgoal clauses, in phase 3 the provers proceed again by tackling the problem in parallel. This proof run stops if one of the provers is able to find a closed tableau or if both provers fail to solve the proof problem within a given time limit.

In order to make the architecture and the behavior more precise, we will describe the used variants of theorem provers and the way subgoal clauses are exchanged in more detail.

We coupled two variants of the powerful prover SETHEO ([LSBB92]). The parameters of SETHEO were chosen automatically as described in [MIL+97]. However, we chose the iterative deepening bounds of the two variants on our own. We let one prover employ the weighted-depth bound because this was the most powerful bound for our problem set. For the second prover we experimented with both the inference bound and the depth bound.

Assigning the depth bound to the second prover is sensible due to different reasons. The most important is that the depth bound is rather powerful when working alone. Thus, it can possibly increase the performance of our whole system. Moreover, as we have investigated in section 4.2.1 the depth bound allows for the generation of subgoal clauses that provide certain guarantees regarding proof length and depth reductions. Since the number of inferences needed to infer a tableau and especially the depth of a tableau influences the resource value of the weighted-depth bound, often the minimal segment of the search tree which contains a proof might be reduced. Hence, a consistent gain of efficiency can be expected.

In contrast, if we combine the weighted-depth bound and the inference bound, we have the problem that a prover based on the inference bound is usually not very powerful when working alone. At least it can support its counterpart. Indeed, since the weighted-depth bound is strongly depth-oriented we often do not have a reduction of the proof search since proof length reductions need not entail depth reductions (see section 4.2.1). But in some cases it is possible to shorten long branches and hence enable a depth-based prover to solve problems that were completely out of reach before. It is to be expected that the inference bound can allow for some high speed-ups but that consistent gains of efficiency are impossible.

The generation and selection process of subgoal clauses orients itself on the principles introduced in section 4. Thus, each prover generates a pool of subgoal clauses with the help of the algorithm from Figure 3. The values Δ_{min} and Δ_{max} were set to 10 and 100, respectively. We did not generate subgoal clauses with context because the results were satisfactory with conventional subgoal clauses. Note that subgoal clauses can be generated without modifying SETHEO. It is only necessary to add some commands of the PROLOG-style input language of SETHEO to the input clauses. Then, in an additional preprocessing before the cooperation phase 2 SETHEO is applied to this modified input set. The output of this preprocessing of SETHEO are the generated subgoal clauses.

For the selection of clauses we fall back on the introduced selection function ρ with $f_{sgc} = f_{sgt} = 1$. The number of selected clauses depends on the problem domain. It can be found in the following subsection. Note that in our system both provers apply the rule goal lift-up to *all selected* subgoal clauses. That is, each prover adds its own selected subgoal clauses and the selected clauses from its counterpart to its input set. This supports the simultaneous search for tableaux of type A and B if the weighted-depth and the inference bound are coupled (see section 4.3.3). If the weighted-depth and the depth bound are coupled both provers can profit from often occurring proof depth reductions stemming from subgoal clauses generated with the depth bound.

5.2 Results

We have conducted experimental studies in the light of several problems taken from the TPTP, v.1.2.1 ([SSY94]). In particular, we have chosen the domains BOO (boolean algebras), COL (combinatory logic), CAT (category theory), and GRP (group theory). The reason for this is that these domains cover a wide range of different problems with different difficulty. In BOO, COL, and GRP there are mainly Horn problems, in CAT also non-Horn problems. Most problems contain equality.

For selecting subgoal clauses we chose following parameters. In the COL and GRP domain we selected 10 subgoal clauses, in the domains CAT and BOO only 5 subgoal clauses. It is to be emphasized that these parameters are not the result of many empirical studies. Hence it might be that our results can be improved.

Table 1 presents the results obtained in our test domains when using SPARCstations 20/712. We only list "hard" problems, i.e. problems which cannot be solved within 10 seconds by any prover employing the depth, weighted-depth, or inference bound. Moreover, we omitted problems that were out of reach of single provers as well as of the cooperative system. All runtimes in Table 1 are given in seconds and include the time for all phases. A "–" denotes that the problem could not be solved within 500 seconds. The table displays in column 1 the name of the problem assigned to it in the TPTP. Column 2 gives the runtime of the best single bound, the weighted-depth bound. Column 3 displays the runtime of a competitive system consisting of provers

Table 1: Combining iterative deepening procedures (1)

problem	wdepth	comp.	goal lift-up	problem	wdepth	comp.	goal lift-up
COL003-3	386.8	60.4	137.2	COL066-1	–	–	210.2
COL003-4	82.3	18.9	41.6	BOO003-2	290.9	290.9	36.1
COL003-5	167.3	167.3	79.5	BOO003-4	18.7	18.7	24.1
COL003-7	66.6	66.6	38.4	BOO004-2	285.5	285.5	62.1
COL003-8	201.8	201.8	380.4	BOO004-4	18.6	18.6	25.1
COL003-9	369.8	27.5	63.8	BOO005-2	384.2	384.2	421.3
COL034-1	166.8	59.6	59.4	BOO005-4	28.8	28.8	33.9
COL036-1	228.5	108.1	104.9	BOO006-2	386.2	386.2	425.9
COL037-1	146.2	110.1	137.6	BOO006-4	29.1	29.1	38.8
COL038-1	275.4	110.3	108.9	BOO009-1	138.3	138.3	8.7
COL041-1	143.7	39.3	43.4	BOO010-1	287.2	287.2	300.3
COL042-2	–	–	209.4	BOO013-1	105.8	22.0	152.5
COL057-1	25.1	12.2	8.2	BOO013-3	215.1	215.1	79.9
COL058-3	123.6	46.3	47.8	BOO016-1	–	–	21.6
COL060-1	14.0	14.0	8.6	CAT004-3	23.3	23.3	22.8
COL060-2	120.6	120.6	5.2	CAT008-1	126.2	126.2	62.2
COL061-1	17.2	17.2	25.1	GRP010-4	87.6	87.6	455.0
COL061-2	80.8	80.8	9.6	GRP124-9.004	261.2	261.2	275.2
COL062-1	–	–	11.8	GRP140-1	–	–	453.8
COL062-2	88.0	88.0	24.0	GRP148-1	–	–	463.5
COL062-3	80.1	80.1	5.7	GRP162-1	44.0	44.0	100.9
COL063-1	–	–	19.4	GRP163-1	115.4	115.4	321.0
COL063-2	99.3	99.3	264.7	GRP165-1	111.9	54.7	12.5
COL063-3	82.2	82.2	104.3	GRP165-2	–	–	310.8
COL063-4	87.4	87.4	13.8	GRP166-3	115.5	115.5	13.4
COL063-5	136.7	136.7	99.5	GRP166-4	–	–	318.0
COL063-6	80.3	80.3	6.4	GRP168-1	150.0	62.9	14.0
COL064-1	–	–	499.1	GRP168-2	156.1	76.9	13.9
COL064-6	83.4	83.4	350.8	GRP186-4	–	–	311.3

employing the depth, inference, and weighted-depth bound. Columns 4 presents the best results when combining the weighted-depth and another bound with goal lift-up and lemma exchange, respectively. In the BOO domain, coupling the weighted-depth and the inference bound led to the best results. In the CAT, COL, and GRP domains, the depth bound was the best partner for the weighted-depth bound.

Table 2 gives an overview on the results. It summarizes the results of Table 1 in form of the number of problems which are solved with each variant and the accumulated runtime. This time is computed as the sum of the runtimes of each problem occurring in Table 1. The runtime of an unsolved problem is counted by 500 seconds.

Both tables show the benefits which can be obtained by the combination of iterative deepening procedures. The success rate of the prover SETHEO can clearly be improved. Whereas only 47 hard problems can be solved by the weighted-depth bound or a co-operative version of SETHEO, the combination of iterative deepening bounds with goal lift-up can raise the success rate up to the value 58. That is, 23.4% more hard problems can be solved. This increase of the success rate concerns the domains BOO, COL, and

Table 2: Combining iterative deepening procedures (2)

		wdepth	competitive	goal lift-up
BOO	solved	12	12	13
	runtime	2688.1	2604.3	1629.9
CAT	solved	2	2	2
	runtime	149.5	149.5	85.0
COL	solved	25	25	30
	runtime	5853.9	4498.1	3118.3
GRP	solved	8	8	13
	runtime	3541.7	3318.2	3063.3

GRP. In CAT at least the same number of problems compared to a competitive prover can be solved. Furthermore, in all domains the accumulated runtime can be decreased. Specifically in the domains COL and BOO proofs can be found very fast.

When taking a closer look at the proofs found by the cooperative system we can recognize the following. As expected, if the weighted-depth bound is coupled with the inference bound merely the weighted-depth bound is able to find proofs. But very often proofs can be found quickly when starting with subgoal clauses generated by the prover based on the inference bound. If the depth bound is a member of the cooperative team also this bound can often solve proof problems when starting with subgoal clauses. In particular in the COL domain, the depth bound is an adequate partner for the weighted-depth bound.

Strong gains of efficiency and the solution of new problems are always the result of resource reductions. Despite the increase of the branching rate through subgoal clauses proofs can be found faster. In only very few cases a resource reduction occurred but a proof could not be found faster (e.g., `COL063-2`, `COL063-3`, `COL064-6`, `BOO003-4`, `BOO004-4`, `BOO005-4`, `GRP162-1`). That is, the selection of only few relevant subgoal clauses indeed usually leads to smaller minimal proof segments. Moreover, the high success rates show that our selection functions really succeed in selecting subgoal clauses entailing resource reductions.

In the cases where no resource reductions took place (either because reduction steps were needed or the wrong subgoal clauses were selected) the use of subgoal clauses did never entail a favorable rearrangement of the search. There, the increase of the branching rate of the search tree always led to higher runtimes than the runtimes of SETHEO (e.g., `BOO006-4`, `GRP010-4`, `GRP124-9.004`). In the most cases these runtimes were merely slightly higher since small numbers of subgoal clauses were selected.

6 Conclusion

We have introduced an approach for combining different connection-tableau-based provers. The approach is developed so as to overcome one main problem of *CTC* provers, namely the inflexibility of the search bounds in use. By combining different iterative deepening procedures different search bounds can support each other and overcome their weaknesses. Combination is achieved by recording subdeductions performed when using certain bounds in the form of clauses. These clauses can be used by provers based on other bounds in order to reorganize their search and find proofs in smaller proof segments.

We identified the well-known technique of lemma exchange ([AS92, Sch94, AL97, Fuc98d]) as a specific instance of our general combination approach. There intermediate solutions of subgoals are stored and exchanged with other provers.

Moreover, we have introduced a second instance of the combination approach: goal lift-up. By goal lift-up recorded transformations of start clauses into subgoal clauses are used. We examined the potential of goal lift-up regarding proof length and proof search reduction and developed heuristic techniques so as to make the approach applicable in practice. Experimental studies conducted with a state-of-the-art prover in the TPTP library reveal the potential of our approach.

There is a related approach that also deals with subgoal clauses. However, subgoal clauses were so far—to our knowledge—not employed for coupling different CTC-based provers. In [Fuc98a] they were used as an additional input of a superposition prover. Because of the different search schemes of CTC and superposition the theoretic potential of this approach is smaller than ours, as discussed in [Fuc98a]. The empirical results presented there demonstrate that—in spite of the theoretical weaknesses—this is also a sensible method.

References

[AL97] O.L. Astrachan and D.W. Loveland. The Use of Lemmas in the Model Elimination Procedure. *Journal of Automated Reasoning*, 19(1):117–141, 1997.

[AS92] O.L. Astrachan and M.E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *Proceedings of CADE-11*, pages 224–238, Saratoga Springs, USA, 1992. Springer, LNAI 607.

[DF94] J. Denzinger and M. Fuchs. Goal oriented equational theorem proving. In *Proceedings of KI-94*, pages 343–354. Springer, LNAI 861, 1994.

[DF98] J. Denzinger and D. Fuchs. Enhancing conventional search systems with multi-agent techniques: a case study. In *Proceedings of ICMAS-98*, pages 419–420. IEEE Computer Society, 1998.

[FD97] D. Fuchs and J. Denzinger. Knowledge-based cooperation between theorem provers by techs. Technical Report SR-97-11, University of Kaiserslautern, Kaiserslautern, 1997.

[Fit96] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, 1996.

[Fuc98a] D. Fuchs. Cooperation of Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer. In *Proceedings of AISC-98*, pages 157–169. Springer, LNAI 1476, 1998.

[Fuc98b] D. Fuchs. Coupling saturation-based provers by exchanging positive/negative information. In *Proceedings of RTA-98*, pages 317–331. Springer, LNCS 1379, 1998.

[Fuc98c] M. Fuchs. Controlled Use of Clausal Lemmas in Connection Tableau Calculi. AR-Report AR-98-02, 1998, Technische Universität München, Institut für Informatik, 1998.

[Fuc98d] M. Fuchs. Relevancy-Based Lemma Selection for Model Elimination using Lazy Tableaux Enumeration. In *Proceedings of ECAI-98*, pages 346–350. John Wiley & Sons, Ltd., 1998.

[Fuc98e] M. Fuchs. Similarity-Based Lemma Generation for Model Elimination. In *Proceedings of CADE-15*, pages 33–37. Springer, LNAI 1421, 1998.

[Har96] J. Harrison. Optimizing proof search in Model Elimination. In *Proceedings of CADE-13*, pages 313–327. Springer, LNAI 1104, 1996.

[Kor85] Richard E. Korf. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *AI*, 27:97 – 109, 1985. Elsevier Publishers B.V. (North-Holland).

[Let98] R. Letz. Using matings for pruning connection tableaux. In *Proceedings of CADE-15*, pages 381–396. LNAI 1421, 1998.

[LMG94] R. Letz, K. Mayr, and C. Goller. Controlled Integration of the Cut Rule into Connection Tableau Calculi. *Journal of Automated Reasoning*, 13:297–337, 1994.

[LSBB92] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. The Model Elimination Provers SETHEO and E-SETHEO. *Journal of Automated Reasoning*, 18(2), 1997.

[Sch94] J. Schumann. Delta - a bottom-up preprocessor for top-down theorem provers. system abstract. In *Proceedings of CADE-12*, pages 774–777. Springer, LNAI 814, 1994.

[Sho76] R.E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *Proceedings of CADE-12*, pages 252–266. LNAI 814, 1994.

[Sti88] M.E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.