Cooperation of Heterogeneous Provers

Jörg Denzinger and **Dirk Fuchs**
Fachbereich Informatik
Universität Kaiserslautern
67663 Kaiserslautern, Germany
{denzinge, dfuchs}@informatik.uni-kl.de

Abstract

We present a methodology for achieving cooperation between already existing theorem provers employing different proof paradigms and/or different search controls, and using different but related logics. Cooperation between the provers is achieved by periodically interchanging clauses which are selected by so-called referees. By employing referees both on the side of a sending prover and a receiving prover the communication is both success- and demand-driven, which results in a rather small communication overhead and synergetical effects.

We report on experiments regarding the cooperation of the provers SPASS, SETHEO and DISCOUNT in domains of the TPTP library and with problems stemming from an application in software component retrieval. The experiments show significant improvements in the number of problems solved as well as in the solution times.

1 Introduction

Search is a central problem solving method employed in AI systems. For most problem areas there exist many systems based on different search paradigms like branch-and-bound, evolutionary algorithms, or problem decomposition. All these paradigms have different strengths and weaknesses and much effort has been invested in implementing systems suitable for dealing with hard and interesting problem instances.

An example for such a problem area is automated deduction. The paradigms used are problem decomposition in analytical provers and (intelligent) enumeration of logical consequences in saturation-based provers. For each paradigm there are several calculi and for each calculus different implementations, typically with many parameters allowing for different incarnations of a prover.

Many experimental studies and prover competitions have shown that it is very difficult to find a priori the best prover for a given problem instance. Therefore competition approaches where several incarnations of one prover or several provers run on different machines until one prover is successful have been proposed (see [Ertel, 1992]) and are used in applications. However, from an economical point of view such approaches are not very efficient, since only one machine contributes to solving the instance. A *cooperation* of the provers would be more interesting and should also lead to finding solutions faster by achieving synergetical effects.

There are approaches for cooperation of homogeneous search systems that have been used for coupling different incarnations of the same prover. Further, Sutcliffe [1992] proposed a concept for heterogeneous provers but failed to deliver convincing results. The problems to solve in order to achieve a successful cooperation of different provers are (1) finding appropriate types of results to interchange, (2) selecting useful results out of the large number of produced results, and (3) having efficient implementations of the provers.

In this paper, we present a general approach for achieving cooperation between search systems using different paradigms that tackles the problems mentioned above for automated deduction. Our TECHS approach (**TE**ams for **C**ooperative **H**eterogeneous **S**earch) allows us to use existing provers with only small modifications, limits the amount of communication by filtering the results of a sending prover, does not unnecessarily disturb the work of receiving provers since received data is filtered with respect to current and future needs, and has the ability to integrate specialized provers.

Our experiments with three state-of-the-art provers, namely SPASS, SETHEO and DISCOUNT, show that their cooperation indeed outperforms the simple competition approach for all tested domains and all time limits. In the tested domains of the TPTP ([Sutcliffe *et al.*, 1994]) TECHS solves up to twice as many hard problems as can be solved by the individual provers taken together.

2 The TECHS Approach

The general idea of the TECHS approach is to interchange selected results between the search agents of a search team in regular time intervals. The search agents are incarnations of the search systems that have been integrated into the distributed system by enhancing them with communication facilities. If there is more than one

incarnation of a search system in the team then the incarnations have to differ in their search control. We call the phases where the search agents independently work on their problem *working phases*, the phases where information is exchanged *cooperation phases*.

To start the team, each agent is given either the whole problem instance to solve or the parts of the instance it can work with. The selection process of results takes place both on the side of a sending agent, by a so-called *send-referee*, and on the side of the receiving agent, by a so-called *receive-referee*. Each search agent may have between 1 and $n-1$ send-referees, if there are n agents in the team, and 1 or no receive-referee.

The task of a send-referee is to select results from the current search state of its agent that should be communicated to the other agents the send-referee is responsible for. The criteria used by a send-referee are either totally based on the success the results have had for its agent or include some general knowledge about the demands of the receiving agents. If a send-referee is responsible for the selection of results for more than one receiving agent, in most cases only success-based criteria can be used. Typically send-referees evaluate results with respect to their syntactic structure and the search steps the results were involved in.

The task of a receive-referee is to select those results sent to its agent that are considered useful for its agent in its current state or in future states. Therefore the criteria used by receive-referees either reflect the current demands and needs of its agent or try to anticipate future demands and needs. Because an agent receives only a small number of results, the criteria used by receive-agents can involve more expensive computations, like the impact of a result on the current search state, than in the case of send-referees.

3 TECHS in Theorem Proving

In theorem proving, the TECHS approach requires different provers running in parallel on different computing nodes. Proof problems are specified in first-order clausal logic. The provers employ either calculi which are complete for first-order logic (*universal provers*) or calculi which are complete for a sub-logic of first-order logic (*specialized provers*). We allow the use of saturation-based and analytic calculi.

In the following, we first introduce the saturation-based calculi *superposition* and *unfailing completion*, and the analytical *connection tableau calculus*. This is because we employ provers based on these calculi in our experiments. After that, we explain in detail how the TECHS concept can be instantiated for provers based on the introduced calculi. Since clauses are to be exchanged between the provers we must at first cope with the topic of how to *extract clauses from the proof search*. After that, we proceed by introducing several methods for *selecting clauses*, i.e. we describe possible send- and receive-referees. Finally, we explain how a prover can *integrate received clauses* into its search state.

3.1 Basics of Theorem Proving

The superposition calculus ([Bachmair and Ganzinger, 1994]) contains several inference rules which can be applied to a set of clauses that constitute a search state. Generally, its inference rules are *expansion* inference rules (superposition left and right, equality resolution, and equality factoring). These rules allow the generation of new clauses that are added to the search state. All inference rules are well-suited for handling equality, i.e. no equality axioms are needed. In addition to expansion rules contraction rules like rewriting or subsumption can be employed. These rules delete redundant clauses or replace clauses by simpler ones.

In order to show the inconsistency of a clause set $\mathcal{C}$, starting with $\mathcal{C}$ derivations of clause sets have to be performed until a set $\mathcal{D}$ is derived which contains the empty clause. A theorem prover based on superposition usually maintains a set $\mathcal{F}^P$ of so-called *passive clauses* from which it selects and removes one clause C at a time. This clause is put into the set $\mathcal{F}^A$ of *activated clauses*. Activated clauses are, unlike passive clauses, allowed to produce new clauses via the application of some inference rules. The inferred new clauses are put into $\mathcal{F}^P$. Initially, $\mathcal{F}^A = \emptyset$ and $\mathcal{F}^P = \mathcal{C}$. The search stops if the empty clause is derived. The indeterministic selection or *activation step* is realized by heuristic means. A heuristic $\mathcal{H}$ associates a number $\omega_C \in \mathbb{N}$ with each $C \in \mathcal{F}^P$, and the $C \in \mathcal{F}^P$ with the smallest *weight* ω_C is selected.

The unfailing completion procedure ([Bachmair *et al.*, 1989]) essentially is a restriction of superposition to unit equations and allows for the development of efficient provers for pure equational logic.

The connection tableau calculus (*CTC* for short) works on *connection tableaux* for a clause set $\mathcal{C}$ (see [Letz *et al.*, 1994]). It employs three inference rules in order to transform one tableau (tree of literals) into another. The *start rule* attaches the literals of an input clause beneath the unmarked root node. Tableau *reduction* unifies a literal s at the leaf of an open branch with the complement of a literal r on the same branch, and applies the substitution to the whole tableau. Hence, by reduction branches can be closed. If a literal of an input clause is unifiable with the complement of the leaf literal of an open branch, *extension* can attach the literals of the input clause beneath the leaf literal. Since the tableau must remain connected after the extension step (each non-leaf literal must be identical to the complement of one of its immediate successors in the tree) the tableau must be instantiated with this unifier (see [Letz *et al.*, 1994]). Because the *CTC* has no specific rules for handling equality, equality axioms must possibly be added to the axiomatization.

In order to show the inconsistency of a set $\mathcal{C}$, all tableaux derivable from the trivial tableau have to be enumerated until a closed tableau appears. This search space can be represented by a search tree. Normally, tableaux enumeration procedures apply *consecutively bounded iterative deepening search with backtracking* ([Korf, 1985]). In this approach iteratively larger

finite initial parts of the search tree are explored with depth-first search.

3.2 Extraction of clauses

The first step of an exchange of information is the extraction of clauses from the search states a prover has produced in a working phase. We distinguish between saturation-based and CTC-based provers.

The extraction of valid clauses from search states of saturation-based provers is very easy because search states are sets of clauses. Even more, a theorem prover can take all interesting clauses to be selected from the *actual search state* at the beginning of a cooperation phase. This is because the provers only derive new clauses, delete unnecessary clauses, or simplify clauses. Hence, the sequence of produced search states (clause sets) is not needed. It is sufficient to consider the most recently derived clause set. We restricted ourselves to the extraction of active clauses since these clauses are maximally contracted w.r.t. the current clause set.

CTC-based provers conduct a search with iterative deepening and backtracking in the search tree of all connection tableaux. A single search state is only one of these tableaux. Hence, it is only possible to extract information on *one* proof attempt from a search state. If the proof attempt represented by the search state does not lead to a proof it might be that all clauses which can be extracted from it are unnecessary. Then, send-referees might have to chose from a pool consisting of unnecessary clauses only. This shows that clauses must be extracted from *all* search states enumerated during a working phase, i.e. they must essentially be extracted from the search tree of all tableaux.

Because of the fact that the search tree of all tableaux is in general not given explicitly, clauses must be extracted from the enumerated tableaux during the search process. A possible method to do this is to utilize *bottom-up lemma mechanisms* (see, e.g., [Letz *et al.*, 1994]) of connection tableau-based theorem provers. We used a second variant and extracted *top-down lemmas*, so-called subgoal clauses (see [Fuchs, 1998a]), from a tableau. A subgoal clause is the clause of the open leaf literals of a tableau. Before a send-referee is applied to the set of extracted clauses it is reasonable to compute a *minimal* clause set by eliminating subsumed clauses.

3.3 Realization of send-referees

A send-referee in a system based on the TECHS approach consists of a pair (S, φ) of a *filter predicate* S and a *selection function* φ. The prover that receives the results of the send-referee obtains those clauses in the cooperation phases that pass through the filter and that are selected by φ. The filter predicate S limits the set of clauses that are eligible for transmission. Typically, clauses are filtered out that are redundant w.r.t. the receivers. Redundant means that a receiver cannot use the clauses in its inference mechanism due to its specific logic. The selection function φ can employ several *judgment functions* $\psi_1, \ldots, \psi_n$. These functions ψ_i associate a natural number with each clause C, and C is considered the better the higher the value $\psi_i(C)$ is. φ eventually selects the clauses with the best judgments.

Firstly, we have developed a judgment function ψ_{gen} which considers *syntactic features of clauses*. It computes a weighted sum of the number of variables and two times the number of function symbols of a clause and hence prefers general short clauses which contain many variables and only few function symbols. These clauses are useful for saturation-based provers because they might often take part in contracting inferences (subsumption, rewriting). They are often also useful for CTC-based provers because open tableau branches might be closed by extending the branches with them and closing the introduced literals with reduction steps.

Secondly, function ψ_{hist} judges the *role of clauses in the search process* conducted so far. It is defined by $\psi_{hist}(C) = \alpha_1 \cdot \mathsf{sub}(C) + \alpha_2 \cdot \mathsf{exp}(C)$, $\alpha_1 > 0 > \alpha_2$. For a saturation-based prover, $\mathsf{sub}(C)$ denotes the number of clauses subsumed by C, $\mathsf{exp}(C)$ is the number of expansion inferences C was involved in. For a CTC-based prover, $\mathsf{sub}(C)$ is the number of clauses subsumed by C after the extraction and $\mathsf{exp}(C)$ is the number of extension or reduction inferences applied to (instances) of literals of C. If the receiver of clauses is a saturation-based prover the idea is to select clauses which may reduce the search space by subsuming other clauses without simultaneously introducing many new clauses by expanding inferences. The criterion is also sensible for receiving CTC-based provers because an uncontrolled increase of the search space caused by the additional clauses might be prevented.

Finally, we employ the function ψ_{deriv} that *judges the derivation tree of a clause* w.r.t. its possible usefulness for the receiver. The derivation tree of a clause taken from a clause set of a saturation-based prover can partially be constructed by tracing back the inferences needed to infer the clause as far as possible (recall that clauses may be deleted during the derivation). The derivation tree of a subgoal clause of a CTC-based prover is represented by the tableau the clause was extracted from. If a clause is selected for a saturation-based prover utilizing heuristic $\mathcal{H}$ the function ψ_{deriv} considers whether clauses with a high heuristic weight regarding $\mathcal{H}$ are part of the derivation tree. If this is the case, it might be difficult for the receiver to infer the clause on its own and hence the proof search can drastically be reduced if the clause is needed in a proof. If the receiver of a clause is a CTC-based prover and the sender a saturation-based prover, we consider whether the clause is the result of an equality inference like superposition. Such clauses are especially useful for CTC-based provers since they have difficulties handling equality. If sender and receiver are CTC-based provers, such subgoal clauses are useful whose respective tableaux can only be inferred by the receiver if a high resource value is employed. Then, high speed-ups can occur if such clauses can be used in a proof.

Due to lack of space we cannot describe the referees in detail. Exact descriptions of send-referees for superposition and unfailing completion provers can be found in [Fuchs and Denzinger, 1997], descriptions of send-referees for CTC-based provers in [Fuchs, 1998a] and [Fuchs, 1999].

3.4 Realization of receive-referees

A receive-referee is a selection function φ which also uses judgment functions for measuring the quality of clauses.

Our first function ψ_{pop} judges whether a received clause may be *part of a proof* that can quickly be found. For a saturation-based prover ψ_{pop} is defined by $\psi_{pop}(C) = \sum_{C' \in \mathcal{F}^A} \nu(C, C')$. ν judges whether both C and C' contribute to a proof. E.g., for superposition-based provers ν judges whether many short clauses can be derived from C and C'. Exact definitions of the function ν for superposition and unfailing completion provers can be found in [Fuchs and Denzinger, 1997]. In our approach, CTC-based provers employ receive-referees only if they receive clauses from saturation-based provers. Then, we define ψ_{pop} as follows. If $\mathcal{S}$ is the set of extracted subgoal clauses of the receiving prover, $\psi_{pop}(C) = \sum_{C' \in \mathcal{S}} \nu(C, C')$. There, ν judges whether C might contribute to close the tableau represented by the subgoal clause C'. The function simply computes the difference of the number of tableau branches which can be closed by C and the number of branches that remain open.

The second judgment function ψ_{dse} prefers clauses which need not be part of a proof but are nevertheless *able to decrease the search effort* in future. We use this function for saturation-based provers only. The function ψ_{dse} produces higher values if many contracting inferences are possible with a clause. Again, exact definitions can be found in [Fuchs and Denzinger, 1997].

3.5 Integration of clauses

The integration of clauses into the search state of a superposition-based prover is very simple since it works on sets of clauses. Thus, a clause can be integrated by adding it to the set of active clauses and performing all inferences possible with it (see [Fuchs, 1998b]).

For the integration of clauses into the search state of a CTC-based prover there exist two possibilities. The first is to add the new clauses to the old axioms and to perform a re-start of the proof run. The second is to avoid a re-start, to add the new clauses to the old axioms, and to continue the search at the current choice point. The first variant has the disadvantage that the whole search of a CTC-based prover is lost and must possibly be repeated. However, the variant has the advantage that it can be implemented very easily. Moreover, in contrast to the second variant it guarantees that during the iterative deepening process a proof can be found in *a minimal segment of the search space*. Since the segments usually grow exponentially, despite the repetition of parts of the work done so far proofs may then be found faster. Therefore, we decided to employ the simple first variant.

4 Experiments

In order to reveal the power of our cooperation approach we conducted experimental studies with the well-known theorem provers SPASS, SETHEO, and DISCOUNT running on different workstations of a prover network. We have chosen two different test areas: on the one hand several domains from the TPTP problem library v.1.2.1., on the other hand problems stemming from an application in software component retrieval. In the following, all time limits or intervals are measured as wall clock time, which means that for the cooperative runs also the time needed for communication is included. Communication was implemented in the simplest way possible, via reading from and writing in files.

4.1 Experiments with TPTP domains

We have chosen the domains BOO, CAT, COL, and GRP as our test domains. These domains cover a large number of problems with different difficulty. Most problems contain unit equations so that the equational prover DISCOUNT can at least obtain parts of the proof problems.

We coupled the provers SPASS, SETHEO, and DISCOUNT in the following way. SPASS and SETHEO run in their standard settings (that were used for the CADE competition CASC-13). Since the DISCOUNT standard heuristic AddWeight is very similar to that of SPASS, we used a goal-oriented heuristic for DISCOUNT as described in [Denzinger and Fuchs, 1994]. This heuristic is not as powerful as the AddWeight heuristic when working alone but is well-suited for cooperation purposes. We allowed for a bidirectional information exchange between SPASS and SETHEO and between SPASS and DISCOUNT every 5 seconds. SETHEO and DISCOUNT did not exchange information. This is because subgoal clauses of SETHEO are mainly non-unit clauses which are useless for DISCOUNT. Further, it is not sensible to add too many equations to the input set of SETHEO because this increases the branching rate of the search tree too much. The referees were parameterized as follows. SPASS and DISCOUNT selected 10 clauses for each other via send-referees, 5 of these clauses were finally selected by their receive-referees. For achieving cooperation between SPASS and SETHEO, their send-referees selected 40 clauses. Then, their receive-referees selected from these clauses 10 clauses in the domains COL and GRP, 30 clauses in the domains CAT and BOO.

Table 1 presents the results obtained in form of the number of hard problems solved within a certain time limit. A problem is 'hard' if none of the provers alone can solve it within 10 seconds. Table 1 displays the results of the single provers, of a competitive parallel system consisting of the three provers (comp), and the results of cooperative systems consisting of SPASS and SETHEO (SP-SE), SPASS and DISCOUNT (SP-DI), and of all provers (all).

In all tested domains, the cooperation of all provers outperforms the competitive approach for all time limits, for domain CAT it even solves twice the number of problems. Even with only one cooperation phase (i.e.

BOO	SP	SE	DI	comp	SP-SE	SP-DI	all
$\leq 10s$	0	0	0	0	2	0	2
$\leq 100s$	0	3	2	5	4	5	9
$\leq 500s$	4	3	2	7	6	8	10
COL	SP	SE	DI	comp	SP-SE	SP-DI	all
$\leq 10s$	0	0	0	0	1	0	1
$\leq 100s$	2	6	0	8	14	2	14
$\leq 500s$	2	10	0	12	17	2	17

CAT	SP	SE	DI	comp	SP-SE	SP-DI	all
$\leq 10s$	0	0	0	0	6	0	6
$\leq 100s$	5	3	0	7	14	5	14
$\leq 500s$	6	4	0	7	14	5	14
GRP	SP	SE	DI	comp	SP-SE	SP-DI	all
$\leq 10s$	0	0	0	0	5	0	7
$\leq 100s$	28	0	2	28	32	33	37
$\leq 500s$	31	1	4	31	39	44	49

Table 1: Experiments with the TPTP library: numbers of hard problems solved

with a $\leq 10s$ time limit) the results are already better than the results of the single provers and the competitive approach. The BOO and GRP domains show that all systems contribute to the improvement in performance.

It should be noted that the performance of DISCOUNT alone (and in comp) is very bad due to the fact that the results of the goal-oriented heuristic are reported. But even if we "cheat" and report the results of AddWeight, the results of comp still are worse than the TECHS cooperation (while the results in CAT and COL would remain unchanged, for BOO 9 and for GRP 43 problems would be solved within the 500 seconds).

4.2 Experiments with software component retrieval

Besides the use of TPTP domains, we have evaluated TECHS with problems stemming from a concrete application, namely software component retrieval. These problems are generated by the NORA/HAMMR tool that tries to retrieve software components with deductive techniques (see [Schumann and Fischer, 1997]). There, we have the following situation. Software components are available in a software library and specified by their pre- and postconditions. Queries are also formulated by pre- and postconditions. In order to find a suitable library component for a given query proof tasks are constructed that check for each library component whether or not it matches the query (via *plug-in compatibility*). We have chosen a library and queries of list processing functions and constructed several proof problems from it. All in all, we tackled 81 provable problems.

In order to solve these problems, we coupled SPASS, DISCOUNT, and SETHEO in the following way. We used all provers in their standard settings. Whereas SPASS and DISCOUNT were coupled in a bidirectional manner as before, we only allowed SPASS to send selected clauses to SETHEO. The integration of subgoal clauses of SETHEO into the search states of SPASS and DISCOUNT decreased the performance. Again, we let the provers cooperate every 5 seconds. The referees were parameterized as for domains CAT and BOO.

Table 2 presents the results obtained in form of the number of problems solved within certain time limits. Again, we observe a significantly better performance by the cooperation of all systems than by the competitive approach. While the competitive approach can merely slightly increase the number of problems solvable in the time limit, all cooperative combinations significantly increase the success rate. Note again that the best performance is achieved if all systems are working together.

5 Related Work

Distribution and cooperation concepts for search systems have mainly been developed as homogeneous concepts. This is also true for automated theorem proving, as several overview paper indicate (see, for example, [Bonacina and Hsiang, 1994]). A concept, that is by many considered heterogeneous, is A-Teams (see [Talukdar *et al.*, 1993]). A-Teams is a shared-memory, data flow architecture like approach in which agents working on the same type of results use the appropriate part of the shared memory as a common search state. So far, no effort has been undertaken to include tree- or graph-based search paradigms in A-Teams. Also, the cooperation of A-Team agents is much closer than the cooperation of TECHS agents, since an A-Teams agent communicates the results of each search step it has performed. This does not allow to use existing search systems as agents easily. In A-Teams there is also no referee concept for limiting communication.

Another concept, that has a certain heterogeneous flavor, is Teamwork [Denzinger, 1995]. Teamwork has been developed to distribute automated deduction [Denzinger *et al.*, 1997] and has some similarities to TECHS. In Teamwork, periodically the work of the search agents is evaluated by referees that use success-based criteria to judge the whole agent and to select good results. In contrast to TECHS, for all agents a new common start state is generated by using the state of the best agent and the selected results of the other agents. So, most of the agents are different incarnations of one search systems. Other agents, called specialists, can only contribute certain results that are used to direct the search of all agents. Due to the common new start state, there is no use of demand-based criteria for selecting results.

A truly heterogeneous search system was presented in [Hogg and Williams, 1993] for solving graph coloring problems. The system consisted of a tree-based search agent that heuristically constructed the possible colorings and a repair agent, that worked on inconsistent colorings in order to generate a consistent one. The agents communicate via a blackboard on which partial color-

T_{max} (secs.)	SPASS	SETHEO	DISCOUNT	competitive	SP-SE	SP-DI	all
10	37	39	0	48	49	37	49
30	47	39	0	54	57	47	57
60	48	45	0	55	57	50	58
120	50	48	0	56	60	54	63

Table 2: Experiments with software component retrieval: numbers of problems solved

ings (called hints) are written in random intervals. The tree-based agent also randomly reads hints and chooses its node that fits to the hint to work on instead of the node its heuristic would choose. Whenever the repair agent is stuck, it uses with a certain probability a hint to generate a new coloring instead of generating a random one. Obviously, a knowledge-based selection using referees is not included in this approach.

In [Sutcliffe, 1992] a heterogeneous cooperation concept for automated deduction was proposed. The central concept was a distributed implementation of a shared memory (a so-called tuple-space), in which each agent wrote all formulas generated by it. No selection process was involved and no existing provers could be used. Therefore the experiments were not convincing.

6 Future Work

We have presented a cooperation methodology for existing search systems and we have demonstrated that this methodology yields significant improvements in the area of automated deduction. Besides testing the methodology for other areas of search (cooperation of genetic algorithms with branch and bound, for example), future work will also be directed to using other types of information than clauses. These other types include selected *negative results* and various *control information*.

Negative results of an agent may be used by other agents to either avoid producing such results or to remove them from their search states. Control information can mostly be used by agents of the same type so as to improve their search focus or to avoid searching in the same parts of the search space. But also agents of different types may gain from certain control information.

References

[Bachmair *et al.*, 1989] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion without Failure. In *Coll. on the Resolution of Equations in Algebraic Structures.* Academic Press, Austin, 1989.

[Bachmair and Ganzinger, 1994] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[Bonacina and Hsiang, 1994] M.P. Bonacina and J. Hsiang. Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning* 13:1-33, 1994.

[Denzinger, 1995] J. Denzinger. Knowledge-Based Distributed Search Using Teamwork. In *Proc. ICMAS-95*, San Francisco, AAAI-Press, 1995, pp. 81–88.

[Denzinger and Fuchs, 1994] J. Denzinger and M. Fuchs. Goal-oriented equational theorem proving using teamwork. In *Proc. 18th KI-94*, Saarbrücken, LNAI 861, 1994, pp. 343–354.

[Denzinger *et al.*, 1997] J. Denzinger, Marc Fuchs, and Matthias Fuchs. High Performance ATP Systems by Combining Several AI Methods. In *Proc. IJCAI-97*, Nagoya, Morgan Kaufmann, 1997, pp. 102–107.

[Ertel, 1992] W. Ertel. OR-parallel theorem proving with random competition. In *Proc. LPAR'92*, St. Petersburg, LNAI 624, 1992, pp. 226–237.

[Fuchs and Denzinger, 1997] D. Fuchs and J. Denzinger. Knowledge-based cooperation between theorem provers by TECHS. *Technical Report SR-97-11*, U. Kaiserslautern, 1997.

[Fuchs, 1998a] D. Fuchs. Cooperation between Top-Down and Bottom-Up Theorem Provers by Subgoal Clause Transfer. In *Proc. AISC-98*, Plattsburgh, NY, USA, LNAI 1476, 1998, pp. 157-169.

[Fuchs, 1998b] D. Fuchs. Coupling Saturation-Based Provers by Exchanging Positive/Negative Information. In *Proc. RTA-98*, Tsukuba, LNCS 1379, 1998, pp. 317-331.

[Fuchs, 1999] D. Fuchs. On the Use of Subgoal Clauses in Bottom-up and Top-down Calculi. *Fundamenta Informaticae*, to appear, 1999.

[Hogg and Williams, 1993] T. Hogg and C.P. Williams. Solving the Really Hard Problems with Cooperative Search. In *Proc. AAAI-93*, Washington, AAAI-Press, 1993, pp. 231–236.

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

[Letz *et al.*, 1994] 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.

[Schumann and Fischer, 1997] Schumann, J.; Fischer, B.: Making Deduction-Based Software Component Retrieval Practical. Proc. 12th Intl. Conf. Automated Software Engineering, Lake Tahoe, 1997, pp. 246–254.

[Sutcliffe, 1992] G. Sutcliffe. A Heterogeneous Parallel Deduction System. In Proc. Workshop on Automated Deduction: Logic Programming and Parallel Computing Approaches, *FGCS'92*, Tokyo, 1992.

[Sutcliffe *et al.*, 1994] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. In *Proc. 12th CADE*, Nancy, LNAI 814, 1994, pp. 252–266.

[Talukdar *et al.*, 1993] S.N. Talukdar, P.S. de Souza, and S. Murthy. Organizations for Computer-based Agents. *Journal of Engineering Intelligent Systems*, 1993.