High Performance ATP Systems by Combining Several AI Methods

Jorg Denzinger
Fachbereich Informatik
Universitat Kaiserslautern
67663 Kaiserslautern
Germany

Marc Fuchs
Fakultat fur Informatik
TU Miinchen
80290 Miinchen
Germany

Matthias Fuchs
Fachbereich Informatik
Universitat Kaiserslautern
67663 Kaiserslautern
Germany

Abstract

We present a design for an automated theorem prover that controls its search based on ideas from several areas of artificial intelligence (AI). The combination of case-based reasoning, several similarity concepts, a cooperation concept of distributed AI and reactive planning enables a system to learn from previous successful proof attempts. In a kind of bootstrapping process easy problems are used to solve more and more complicated ones. We provide case studies from two domains in pure equational theorem proving. These case studies show that an instantiation of our architecture achieves a high grade of automation and outperforms state-of-the-art conventional theorem provers.

1 Introduction

Research concerned with achieving more efficient (fully automated) theorem provers focuses on three directions: higher inference rates, eliminating unnecessary inferences, and better control of the search. Although all these directions can indeed lead to more efficiency, better control of the search offers the highest gains, but also causes the most problems and has some risks. Nearly all approaches to improving search control involve the use of techniques and methods from other areas of artificial intelligence (AI), as for example knowledge representation, case-based reasoning (CBR), learning, planning, or multi-agent systems. In most of the known works only ideas from one of these areas are exploited.

One area' that should—from the human point of view—be the most promising for high efficiency gains is learning. But the use of machine-learning techniques for improving automated theorem provers faces several severe problems. Learned knowledge has to be stored, retrieved, and very often must be combined. So, the focus of attention should not be restricted to the area of machine learning. Other areas of AI must also contribute in order to successfully apply the results the learning techniques produce.

In this paper we present an approach to controlling the search of an automated theorem prover that combines techniques from several areas of AI to overcome the problems that arise when trying to learn and to use control knowledge. The central idea is to utilize a known (i.e., learned) proof of a so-called *source problem* solved previously in order to guide the search for a proof of the *target problem* at hand. To this end we employ a method *called flexible re-enactment* (cp. [9]).

Source problems must of course satisfy certain similarity criteria with respect to the target problem. Our techniques for maintaining a database of source problems and our mechanisms for selecting source problems that are the most similar to the target are inspired by CBR. Unfortunately, one of the important premises of CBR, namely that "smail *differences between problems result in small differences of their solutions",* is not fulfilled in automated theorem proving.

We cope with this uncertainty by applying the TEAMWORK method ([4]), a multi-agent approach to distributed search. TEAMWORK reduces the risk of deploying an inappropriate heuristic by having a *team* of heuristics (agents) guide the search *concurrently* and *cooperatively.* The reactive planning capabilities of a further agent, namely the supervisor, are made use of to compose a suitable team (cf. [7]). Moreover, the selection of the most suitable source problem required by flexible re-enactment can also be integrated with TEAMWORK in form of a specialized agent.

The combination of all these AI methods allowed us to build a theorem prover for pure equality reasoning that is fully automated, in both learning and proving, and is able to solve hard problems by using a kind of bootstrapping process that starts with easy problems and uses their proofs to gradually solve harder and harder problems. Besides providing the problems, no interaction with the system is required. Our experiments validate and substantiate the achievements of our system.

We cannot provide as many details as some readers (and we) would like. These readers may refer to [6].

2 Equational Reasoning

Equational reasoning deals with the following problem: Given a finite set *E* of equations (of terms over a fixed signature *sig)* and a goal *u = v.* The question is whether the goal equation is a logical consequence of E, i.e., *E* (=

$u = v$. Unfailing completion (e.g., [2]) has proven to be quite successful for solving such a *proof problem* $A = (E, u = v)$. The method is also a good example for so-called generating calculi that are based on generating new facts until a fact describing the goal is reached.

The inference rules of a generating theorem prover can be divided into two classes: *expansion* and *contraction* rules (see [3]). Completion uses the expansion *critical-pair-generation* and the contractions *reduction* and *subsumption.* Basis for the completion procedure is a so-called reduction ordering y that is used to restrict the applicability of the inference rules and to avoid cycles.

An algorithmic realization of the inference rules of a generating theorem prover can be characterized as follows. There are two sets of facts (equations in our case): the set F^A of *active facts* and the set F^p *of passive facts.* The algorithm centers on a main loop with the following body: At first a fact A is selected and removed from F^p ("activate A"). After that A is normalized resulting in a fact A'. (Normalization denotes the application of contraction rules to a fact until none of these rules is applicable anymore.) If A' is neither trivial nor subsumed by an active fact, all elements of F^A are normalized, A' is added to F^A, and all facts that can be generated with A' and other elements of F^A are added to F^p. *A* proof is found if the normalization of the two terms of the goal leads to the same term.

Assuming that there is a given order in which contracting inference rules are applied, normalization of a fact is a deterministic process. Hence, the remaining indeterminism is to determine which fact should be activated next. In order to eliminate this indeterminism, selection strategies and heuristics are used (see, e.g., [5]). In section 4 we present such a selection heuristic that is based on re-enacting a successful proof attempt for a problem that is somewhat *similar* to the problem at hand.

Since we want to learn from (successful) proof attempts, we have to obtain, represent, and store an actual proof run produced by the algorithm and a selection heuristic. SA denotes the sequence of facts activated during a proof run for problem A using a fixed heuristic H. The actual proof to A is denoted by PA and it is obtained by eliminating from SA all facts that did not contribute to the proof. We refer to the facts occurring in PA also as the set PA of *positive facts.* The other facts in SA form the set NA of negative facts that is needed for some learning approaches (see [8; 9]). A successful proof attempt is stored as the quadruple (A,H,S_A,V_A) (or $\{A,C,S_A,VA)$ with C denoting the teams used) that allows the use of various approaches for learning from previous proof experience.

3 Teamwork

The TEAMWORK method is a knowledge-based distribution method for certain search processes ([4]). Equational deduction by completion, as well as for example first-order deduction by (hyper-) resolution, is a member of this class of search processes. In a TEAMWORK-based system there are four different types of agents: experts, specialists, referees, and a supervisor. Experts and specialists are the agents that work on really solving a given problem. *Experts* form the core of a team. They are problem solvers (in our case theorem pro vers) that use the same inference mechanism (in our case unfailing completion), but different selection strategies for the next inference step to do. *Specialists* can also search for a solution (using other inference mechanisms) or they can help the supervisor, for example by analyzing and classifying the given problem like PES (see section 5.2). Each expert/specialist needs its own computing node. Therefore, the supervisor determines the subset of experts/specialists that are active during a working period.

After a working period a *team meeting* takes place. In the *judgment phase,* each active expert and specialist is evaluated by a referee. Each referee has two tasks: judging the whole work of the expert/specialist of the last working period and selecting outstanding results. The first task results in a *measure of success,* an objective measure that allows the supervisor to compare the experts. The second task is responsible for the cooperation of the experts and specialists, since each selected result will be part of the common start search state of the next working period. The referees send the results of their work to the supervisor.

In the *cooperation phase* the supervisor has to construct a new starting state for the next working period, select the members of the team for this next period and determine the length of the period. The new start state for the whole team consists of the whole search state of the best expert enriched by the selected results of the other experts and the specialists. The supervisor determines the next team with a reactive planning process involving general information about components and problem domains *(long-term memory)* and actual information about the performance of the components *(short-term memory).* The long-term memory suggests a *plan skeleton* that contains several small teams for different phases of a proof attempt. These suggested teams are reinforced with appropriate experts/specialists (if more computing nodes are available). During each team meeting the plan has to be updated. This means that adjustments are made according to the actual results (see [7]).

TEAMWORK allows for synergetic effects that result in enormous speed-ups and in finding solutions to problems that are beyond the possibilities of the single experts and specialists. While the competition of the experts directs the whole team into interesting (and promising) parts of the search space, the cooperation provides the experts with excellent facts they are not able to come up with alone. Thus gaps in their derivations towards the goal can be closed. This makes TEAMWORK the ideal basis for a learning theorem prover.

4 Flexible Re-enactment

Similarity between two proof problems A and B can occur in many variations. One possible kind of similarity is

that a considerable number of the facts that contribute to a proof of *A* are also useful for proving *B* (or vice versa). This means in our terminology that the associated sets of positive facts *PA* and *PB* or the proofs P*A* and P*B* "have a lot in common" or, in other words, share many facts. *("PA PB PB* is almost equal to *PA* and *PB")* Our goal is to think up a heuristic that is able to exploit such a similarity.

Given 2 = *(A$_S$,'H,S$_{AS}$)'P$_{AS}$) AS PAST* experience regarding a source problem *As,* and assuming that a target problem *AT* is similar to *As* in the way just described, it is reasonable to concentrate on mainly deducing facts when attempting to prove *AT* that also played a role in finding the source proof P$_{AS}$ » namely the positive facts *P$_{AS}$* We therefore design a heuristic FlexRE which—when trying to prove *AT*—makes use of J by giving preference to facts that were important for finding P$_{AS}$ Such facts will henceforth be referred to as *focus facts.* Note that focus facts are facts inferred or. inferable in connection with *AT•* They must be distinguished from the positive facts *P$_{AS}$* belonging to the source problem *As,* since it might be the case that some A 6 *P$_{AS}$* is not deducible at all in connection with *AT* (due to a different axiomatization). *P$_{AS}$* is merely used to determine if some fact A inferable in connection with *AT* is a focus fact. To put it another way, the use of *P$_{AS}$* is effected by FlexRE on a strictly heuristic basis, meaning that *P$_{AS}$ only* influences the selection of facts from *F^p*, not, for instance, *F^p* itself. That is, *P$_{AS}$* is a guideline that FlexRE tries to follow if possible.

Depending on how strongly focus facts are preferred, FlexRE will re-enact (parts of) P$_{AS}$ more or less quickly. Some of the focus facts, though useful for proving *As,* may be irrelevant regarding the proof *V$_{AT}$ O*f A$_T$ eventually found. But these irrelevant focus facts are not a big problem. The crucial difficulty is to find those (non-focus) facts that have to supplement the relevant focus facts in order to obtain a proof *V$_{AT}$•* It is very likely that these (few) missing facts are descendants of relevant focus facts. Consequently, FlexRE should also favor descendants of focus facts. Favoring descendants should weaken with their "distance" from focus facts, since it cannot be assumed that the few missing facts are located very deeply relative to focus facts.[1]

Preferring descendants of focus facts in addition to giving preference to focus facts themselves justifies the attribute 'flexible' in the term 'flexible re-enactment' which summarizes the working method of FlexRE (see [9] or [6]).

5 Learning and CBR in the Teamwork Environment

In sections 3 and 4 we concentrated on how to use knowledge learned from previous successful proofs (in form of flexible re-enactment) and on how to overcome the problems such a use might cause (in form of the TEAMWORK method with cooperation with other experts, assessment of experts and results, and reactive planning to adapt to the problem at hand using long- and short-term memory). The problems that remain are how to find a proof that should be re-enacted in order to solve a given target problem and how to structure, build, and maintain the long-term memory from proof run to proof run.

The first problem will be tackled by a specialist PES that is providing the supervisor with information about known proof problems that are similar to the given target problem (see subsection 5.2). The second problem naturally depends on how the proof problems are presented to the system. Found proofs have simply to be extracted, analyzed and stored (the latter depending on how specialist PES will perform its retrieval). As we shall see in the next subsection, the necessary components are already provided in form of TEAMWORK agents.

5.1 The Basic Learning Cycle

Systems that use learning techniques for solving their tasks can be (very) roughly divided into two groups: systems that have a clearly defined learning phase after which (usually) no further learning takes place, and systems that always learn. In automated theorem proving, systems of the first type may be usable in clearly defined situations (see, for example, [8]), but in general learning should never stop.

Nevertheless, one can observe times in the use of a (learning) theorem prover in which new domains are explored, and other times in which one is interested in proving one particular problem. When exploring a new domain, typically there is a set of problems to be solved, and when starting the exploration no knowledge in the prover will be triggered. In the following, we will first concentrate on the exploration of a new domain and then we will point out how the one-problem case is handled.

When exploring a new domain the ordering of the problems given to a prover may influence its success. In order to deal with this problem we decided to let the prover handle the ordering of the given set of problems and also allow the prover to make several attempts to solve a problem. The latter is necessary since each solved problem may result in new knowledge that allows for solving some other problems that could not be solved so far *("bootstrapping").* Note that the set of problems given to the prover has to include easy and typical problems of a domain that the prover can use to get fundamental knowledge about this domain.

As already stated, in a TEAMWORK-based system the long-term memory that represents knowledge about domains is the responsibility of the supervisor. When confronted with a set of example problems of a new domain, the supervisor controls not only the single proof attempts, but a whole series of proof attempts that are to result in solving as many of the problems as possible.

Since the supervisor has no appropriate information when being confronted with a new domain, the first step

[1] "Distance" and (relative) depth basically refer to the number of inference steps separating two facts, one of these facts contributing to the deduction of the other.

is to try to solve the given problems with conventional means, i.e., without the use of experts and specialists that employ learned knowledge. This is accomplished by using a pre-defined team for a few cycles for each of the given problems (in a separate run). After generating $\mathcal{I} = (\mathcal{A}, \mathcal{C}, \mathcal{S}_{\mathcal{A}}, \mathcal{P}_{\mathcal{A}})$ for each solved problem $\mathcal{A}$, this data is integrated into a database of past proof experience that is part of the long-term memory. This database is essentially organized as case base. Hence, the structure and retrieval processes regarding this database are strongly related to CBR techniques (cp. [10] and subsection 5.2). Then the supervisor tries to solve the remaining problems (again imposing a time limit on each run), but now the teams are different. The team of the first working period is again pre-defined, and contains, besides good general purpose experts, the specialist PES.

After the first working period, the supervisor uses its reactive planning process to adapt the team to the problem. If PES was not able to report to the supervisor a problem from the case base that is similar to the target problem at hand, then the supervisor proceeds according to its standard procedure. Otherwise, the expert FlexRE will become a member of the next team, utilizing the reported problem from the case base as source problem.[2] Problems that can now be solved are analyzed so as to produce new data for the case base. For the remaining unsolved problems this whole process is repeated until no more new problems can be proven. Note that after the initial round that uses the pre-defined team without components using learned knowledge, in all other rounds each solved problem is immediately added to the case base so that it already can be used for the proof attempt of the next problem. Thus the number of rounds is often reduced.

In the one-problem case, i.e., in the presence of fundamental domain knowledge, the supervisor immediately employs a team that includes PES, and the supervisor plans and controls the whole proof attempt as described above. If the system is successful, the data of the run is also added to the case base.

5.2 Specialist PES

As described before, specialist PES retrieves one or possibly several source problems that are similar to a given target problem $\mathcal{A}_T$, and transmits information on these problems to the supervisor. More exactly, after receiving a target problem $\mathcal{A}_T = (Ax_T, \lambda_T)$ given over a fixed signature sig_T, PES returns information $R_{\mathbf{PES}}(\mathcal{A}_T)$ on similar problems $\mathcal{A}_S$, where $R_{\mathbf{PES}}(\mathcal{A}_T) = \{(\sigma, P_S) : cond_T(\sigma(\mathcal{A}_S))\}$. The data is determined by $cond_T$ and comprises the set of positive facts P_S associated with source problem $\mathcal{A}_S$, and a signature match σ from $\mathcal{A}_S$ to $\mathcal{A}_T$. (σ provides an appropriate renaming of the function symbols occurring in sig_S.) $cond_T$ denotes that a problem $\mathcal{A}_S$ (translated from sig_S to sig_T by applying σ) is most similar to $\mathcal{A}_T$ (there are no other problems more similar to $\mathcal{A}_T$ than $\mathcal{A}_S$), but also that the similarity between $\mathcal{A}_S$ and $\mathcal{A}_T$ seems to be sufficient.

In order to construct such a predicate we shall use a quasi-ordering $\succeq_T$ that allows us to compare the similarity between proof problems and a target problem. It should hold true that $\mathcal{A}_1 \succ_T \mathcal{A}_2$ if (and only if) $\mathcal{A}_T$ is more similar to $\mathcal{A}_1$ than to $\mathcal{A}_2$. Furthermore, an absolute measure of similarity is needed in order to construct a *minimal similarity* predicate ms that estimates whether the similarity between the target and a source problem seems to be sufficient or not.

Basically, there are two possibilities to construct $\succeq_T$ in order to estimate whether a problem $\mathcal{A}_1$ can provide a more suitable source proof than a problem $\mathcal{A}_2$: On the one hand it is possible to compare the problem descriptions, and on the other hand one can compare the search effort spent on solving the two problems. Considering the working method of FlexRE it is reasonable to consider $\mathcal{A}_1$ as providing a more suitable source proof than $\mathcal{A}_2$, if the target $\mathcal{A}_T$ is more similar to problem $\mathcal{A}_1$ than to $\mathcal{A}_2$ w.r.t. its problem description. Nevertheless some information on the search conducted to solve a problem, namely the length of the search protocol $\mathcal{S}$, can be consulted. Since some of the positive facts of more difficult problems (problems with a longer search protocol) were quite difficult to reach it seems to be sensible to force the activation of these probably useful, but hard to reach facts by using them as focus facts (cp. section 4). Thus more difficult problems should be favored.

Similarity between problem descriptions is surely also suitable for constructing an absolute similarity value in order to decide in favor of or against sufficient similarity. Information on the search protocol is difficult to use for this particular purpose and should only be used as a criterion to compare different problems.

In order to assess the differences between the problem descriptions of two proof problems we employ the similarity measure sim_T. As discussed before, this measure is useful to construct $\succeq_T$ and ms. The design of sim_T is motivated by the fact that a target problem $\mathcal{A}_T = (Ax_T, \lambda_T)$ is proved by virtue of a proof of a source problem $\mathcal{A}_S = (Ax_S, \lambda_S)$, if all axioms of Ax_S are subsumed by axioms of Ax_T and the target goal λ_T is subsumed by the source goal λ_S. Hence, subsumption criteria will play the major role in our approach. Furthermore, refinements by using other criteria (e.g., subsumption modulo AC) are imaginable (see below). These refinements have proven their usefulness in experiments, although naturally a simple proof replay often is impossible if Ax_S and Ax_T are similar in such a way.

In order to realize measure sim_T we introduce a (asymmetric) similarity rating sim_T^{eq} defined on equations over sig_T. Let ax_1 and ax_2 be two equations. We define $sim_T^{eq}(ax_1, ax_2) \in [0;1]$ by $sim_T^{eq}(ax_1, ax_2) = \max\{rating(\rho_i) : \rho_i(ax_1, ax_2),\ 1 \leq i \leq m\}$. $\rho_1, \ldots, \rho_m$

[2] If PES provides more than one possible source problem, then the supervisor can react in various ways: It can select the most similar source problem and discard all others (which was done in section 6), or it may use several experts FlexRE, each using a different source problem, or it may supply the single FlexRE with the source problems provided by PES in succession.

are similarity criteria defined on equations. The function *rating* judges the reliability of a measure in order to judge similarity between two equations.

Possible similarity criteria are the relations $\lhd$, $\lhd_A$, and $\lhd^H \cup \rhd^H$ where $\lhd$ denotes "plain" subsumption, and $\lhd_A$ subsumption modulo the theory given by a set of equations A. (Here, we shall always use $A = AC$). $ax_1 \lhd^H ax_2$ stands for a homeomorphic embedding of ax_2 in ax_1. We used the ratings 1, 0.8, and 0.2 for the criteria $\lhd$, $\lhd_A$, and $\lhd^H \cup \rhd^H$, respectively. Hence, subsumption is considered to be very important, whereas homeomorphic embedding is considered to be a very weak similarity criterion. Note that we actually refined sim_T^{eq} by adding further similarity criteria thus increasing the ability to produce distinctive measures. But a description of these technical details is beyond the scope of this paper.

With the help of this measure we are able to construct a similarity measure defined on proof problems. In the following let $\mathcal{A}_T = (Ax_T, \lambda_T)$ be a target problem, and $\mathcal{A}_S = (Ax_S, \lambda_S)$ be a source problem given over sig_T. Let $Ax_T = \{ax_1, \ldots, ax_m\}$, $Ax_S = \{ax'_1, \ldots, ax'_n\}$ $(n, m > 0)$. The similarity of target and source problem is $sim_T(\mathcal{A}_T, \mathcal{A}_S) = (s_1, s_2, s_3) \in [0;1]^3$, where

$$s_1 = \frac{1}{n} \cdot \sum_{i=1}^{n} \max\{sim_T^{eq}(ax, ax'_i) : ax \in Ax_T\}$$

$$s_2 = \frac{1}{m} \cdot \sum_{i=1}^{m} \max\{sim_T^{eq}(ax_i, ax) : ax \in Ax_S\}$$

$$s_3 = sim_T^{eq}(\lambda_S, \lambda_T)$$

Thus, s_1 judges the degree of "coverage" of Ax_S through similar axioms of Ax_T. For example, we have $s_1 = 1$ if all source axioms are subsumed by target axioms. s_1 decreases if only weaker similarity criteria are fulfilled. The value s_2 represents the percentage of target axioms that have a similar counterpart in Ax_S. Additional axioms in Ax_T do not prevent the source proof from being applicable (in the case $s_1 = 1$), but may complicate the search for this proof. Finally, s_3 measures the similarity between target and source goal λ_T and λ_S. We have $s_3 = 1$ if λ_S subsumes λ_T.

With the help of sim_T we are now able to estimate if the similarity between $\mathcal{A}_T$ and $\mathcal{A}_S$ seems to be sufficient. To this end, we check if the predicate ms is fulfilled with $ms(sim_T(\mathcal{A}_T, \mathcal{A}_S))$ iff $c_1 \cdot s_1 + c_2 \cdot s_2 + c_3 \cdot s_3 \geq min$, where $c_1, c_2, c_3 \in \mathbb{R}$, and $min \in \mathbb{R}$ is the threshold. In our implementation we use $c_1 = 3$, $c_2 = 1$, $c_3 = 2$, and $min = 1$. Hence additional axioms in Ax_T are considered quite harmless, while a good coverage of Ax_S is considered to be important. Since we use $min = 1$, a subsumption of one third of the axioms of Ax_S, or no superfluous target axioms, or a subsumed target goal each suffice alone to reach the threshold.

sim_T is also employed to define $\succeq_T$. We define $\succeq_T$ as a lexicographic combination of two quasi-orderings $\geq_S$ and $\geq_D$, where $\geq_S$ compares the similarity of two problem descriptions with respect to the target problem (using sim_T), and $\geq_D$ compares the length of the search protocols (cp. [6]). Using $\succeq_T$ and ms, we can define $cond_T$. We apply

$$cond_T(\mathcal{A}_S) \quad \text{iff} \quad ms(sim_T(\mathcal{A}_T, \mathcal{A}_S)) \wedge \neg\exists \mathcal{A} : \mathcal{A} \succ_T \mathcal{A}_S.$$

In the subsequent section we shall discuss the experimental results we obtained with DISCOUNT by using specialist PES and expert FlexRE.

6 Experimental Results

Finding appropriate test sets of problems for our learning prover was not easy, because in most publications only hard problems are given. But we must solve at least one problem with conventional means in order to start the bootstrapping process. Fortunately, the TPTP library ([13]) contains two domains that include related problems of various degrees of difficulty. While we could use the domain groups (GRP) without any changes, the domain logic calculi (LCL) consists of several subdomains (some of which contain only hard problems, again) and is not given in a pure unit-equality axiomatization. Therefore we had to transform the problems of one subdomain (the CN calculus) for our experiments.

The achievements of a learning approach can only be observed when results from conventional provers are also provided. Besides two of our experts (AddWeight, or Add for short, and Occnest, see [5]) we will also use OTTER (version 3.0, using the *autonomous mode,* see [11]) to allow a comparison. Since OTTER has won the CADE-13 theorem prover competition ([12]) in the category "Unit Equality Problems", we think that a comparison with the current state-of-the-art is thus provided.

Our learning approach is integrated into our prover DISCOUNT (see [1]), that is implemented in C on Unix machines and has an old and slow inference engine. We limited each distributed run of it to 3 minutes, while the other provers had 10 minutes. The ordering in which DISCOUNT tried the problems is the lexicographical ordering of the names of the problem. All experiments were performed on SUN Sparc-10 machines, the team runs employing two of them. There are 125 problems in the GRP domain and 24 in the LCL (CN) domain (for the transformation into equality problems, for more results and a broader analysis of them, see [6]).

Table 1 shows that our main goal, developing a prover that is able to automatically learn and therefore to solve more problems than other provers, has definitely been achieved. In both domains, our learning team clearly outperformed OTTER and the single experts by at least 15 percent. Table 2 highlights some of the problem solution chains that are produced by our prover. In the GRP domain, problem 179-1 was solved conventionally and used in the next round to solve problem 179-2, which was basis for solving problem 183-1 (in the same round). Using problem 183-1 DISCOUNT was able to solve problem 167-3, which was then used to solve problem 167-1. In the LCL domain, problem 047-1 was used to solve 048-1 which allows for solving 050-1 that then is the source for problem 051-1. Due to the immediate usage of solved

Table 1: Comparison learning team vs. OTTER vs. best experts

Domain	number of problems	learning team		OTTER		Add		Occnest	
		# solved	%	# solved	%	# solved	%	# solved	%
GRP	125	113	90	93	74	86	67	91	73
LCL (CN)	24	17	71	11	46	12	50	6	25

Table 2: Selection of results

target	source	runtime	OTTER	Add	Occnest
179-1	—	12s	—	—	—
179-2	179-1	37s	—	—	—
183-1	179-2	40s	—	—	—
167-3	183-1	129s	—	—	—
167-1	167-3	32s	—	—	—
047-1	—	21s	137s	27s	—
048-1	047-1	14s	138s	27s	35s
050-1	048-1	32s	—	—	—
051-1	050-1	15s	—	—	—

problems when trying to solve the next one, the chain of the GRP domain is the result of 4 rounds, while the chain of the LCL domain was produced in only 3 rounds of the bootstrapping process. Note that any other ordering of the problems would produce the same Table 1 and the same chains in Table 2. Only the number of rounds needed may be different.

Among all the problems there is only one problem that OTTER can solve and our learning team cannot, but 27 problems that our learning team can solve and OTTER cannot.

In general, our experiments show that our concept of a learning theorem prover clearly outperforms current conventional theorem provers if the learning prover is provided with enough "exercise" in the domain it has to work in. In this case even the learning process is accomplished by the prover without help from the user.

7 Conclusion and Future Work

We presented a concept for a learning theorem prover that uses methods from several areas of AI. AI methods like planning and CBR are combined with the TEAMWORK multi-agent architecture, resulting in a theorem prover that clearly outperforms renowned provers. A prerequisite for the success of our system- as for a human student—is the presentation of problem domains in a "learnable" way, meaning that the presented problems cover the whole spectrum of difficulty ranging from easy to challenging. In a kind of bootstrapping process the system is able to solve harder and harder problems without any interaction on the parts of a user ("teacher").

Despite the success of our system, nevertheless most components are only first ideas that leave much room for improvements and extensions. Also, there are other concepts that can be used (e.g., [8]) or are at least worth investigating (e.g., the division of problems into easier sub-problems on the basis of learning), that will provide a wider range in the use of learned knowledge.

Acknowledgments

This work was supported by the *Schwerpunktprogramm Deduktion* of the *Deutsche Forschungsgemeinschaft (DFG).*

References

[I] Avenhaus, J.; Denzinger, J.; Fuchs, M.: DISCOUNT: A System For Distributed Equational Deduction, *Proc. 6th RTA,* Kaiserslautern, LNCS 914, 1995, pp. 397-402.

[2] Bachmair, L.; Dershowitz, N.; Plaisted, D.: Completion without Failure, *Coll. on the Resolution of Equations in Algebraic Structures,* Austin (1987), Academic Press, 1989.

[3] Dershowitz, N.: A maximal-Literal Unit Strategy for Horn Clauses, *Proc. 2nd CTRS,* Montreal, LNCS 516, 1990, pp. 14-25.

[4] Denzinger, J.: Knowledge-Based Distributed Search Using Teamwork, *Proc. ICMAS-95,* San Francisco, AAA1-Press, 1995, pp. 81-88.

[5] Denzinger, J.; Fuchs, M.: Goal-oriented equational theorem proving using teamwork, *Proc. 18th KI-94,* Saarbriicken, LNAI 861, 1994, pp. 343-354.

[6] Denzinger, J.; Fuchs, Marc; Fuchs, M.: High Performance ATP Systems by Combining Several AI Methods, SEKI-Report SR-96-09, University of Kaiserslautern, 1996. (ftp://ftp.uni-kl.de/reports.uni-kl/computer-science/SEKI/ 1996/Denzinger. SR-96-09. ps.gz)

[7] Denzinger, J.; Kronenburg, M.: Planning for Distributed Theorem Proving: The Teamwork Approach, *Proc. KI-96,* Dresden, LNAI 1137, 1996, pp. 43-56.

[8] Fuchs, M.: Learning proof heuristics by adapting parameters, *Proc. 12th WML,* Tahoe City, CA, USA, 1995, pp. 235-243.

[9] Fuchs, M.: Experiments in the Heuristic Use of Past Proof Experience, *Proc. CADE-IS,* New Brunswick, LNAI 1104, 1996, pp. 523-537.

[10] Kolodner, J.L.: An Introduction to Case-Based Reasoning, *Artificial Intelligence Review* 6:3-34, 1992,

[II] McCune, W.W.: OTTER 3.0 Reference manual and Guide, Tech. rep. ANL-94/6, Argonne National Laboratory, 1994.

[12] Sutcliffe, G.; Suttner, C.B.: The Design of the CADE-13 ATP System Competition, *Proc. CADE-13,* New Brunswick, LNAI 1104, 1996, pp. 146-160.

[13] Sutcliffe, G.; Suttner, C.B.; Yemenis, T.: The TPTP Problem Library, *Proc. 12th CADE,* Nancy, LNAI 814, 1994, pp. 252-266.