Planning for distributed theorem proving: The team work approach

Jörg Denzinger, Martin Kronenburg
Department of Computer Science
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Email: {denzinge, kronburg}@informatik.uni-kl.de

Abstract

This paper presents a new way to use planning in automated theorem proving by means of distribution. To overcome the problem that often subtasks for a proof problem can not be detected a priori (which prevents the use of the known planning and distribution techniques) we use a team of experts that work independently with different heuristics on the problem. After a certain amount of time referees judge their results using the impact of the results on the behaviour of the expert and a supervisor combines the selected results to a new starting point.

This supervisor also selects the experts that can work on the problem in the next round. This selection is a reactive planning task. We outline which information the supervisor can use to fulfill this task and how this information is processed to result in a plan or to revise a plan. We also show that the use of planning for the assignment of experts to the team allows the system to solve many different examples in an acceptable time with the same start configuration and without any consultation of the user.

Plans are always subject to change
Shin'a'in proverb

1 Introduction

A main problem of automated theorem proving is the immense search space that even for small problems a theorem prover has to deal with. Research for solutions to this problem centers on two directions, the use of distributed provers and the use of knowledge to guide the prover through the search space. Although this guidance is used quite some time in provers by means of heuristics, only in the last few years better concepts of guidance, namely planning, find their way into proving systems.

Both, distributed provers and provers using planning have to deal with the same problem: the inability of finding a priori appropriate subtasks for a proof problem. Only if very much about a proof problem is known, such subtasks can be determined. But what when the knowledge about the proof problem in addition to the input is vague ?

To deal with this problem we developed the team work method (see [De93], [AD93]) for the distribution of problem solving tasks. It is applicable where the descriptions of the tasks show no obvious ways for distribution. The general idea of team work is to let a team of several so called experts work independently on the proof problem. They differ in the heuristics they use to determine the next step to do. After a given amount of time the experts stop their work and for each expert a referee assesses the work of the expert and reports a general assessment and a few good results to a supervisor. This supervisor collects the reports, generates based on these reports a new starting point for the experts and selects experts and referees for the next working round. We call the phase when the referees and the supervisor work a team meeting.

Structure and behaviour of our teams lead to a system that uses both competition and cooperation between its components (mainly the experts) to solve given problems. The experts compete with each other in order to stay in the team. But with the help of the referees they also cooperate, because their best results are used to form a new starting point for the work of the team.

In [AD93] we mainly concentrated on the general aspects and the theory of this approach and showed that for many examples there exist combinations of experts that allow big speed-ups. In [DF94] we designed powerful experts and demonstrated that we are even able to solve examples using team work that could not be solved by any of our experts working alone. But the question remains how to choose automatically good teams for a given example, so that in general different examples can be solved in an acceptable time. The solution to this problem is the supervisor. During the team meetings he selects the members of the team for the next round.

We see this process of creating a new team in each team meeting as a kind of reactive planning in which the supervisor uses general knowledge about the experts, the referees and their relationships, dependencies and incompatibilities (the long-term memory) in combination with the results of the experts on the given problem so far (the short-term memory), to determine the team members. He selects not only the team members of the next round but also can make assignments for further rounds provided that the selected team comes up with the expected results. Otherwise the behaviour of the selected team is used to find better suited experts. This can be seen as replanning or plan revision.

By planning the team we contribute to the problem of finding subtasks for a proof problem. If subtasks can be detected, they can be assigned to experts that are capable to prove them. If no subtasks can be found, then the supervisor tries several experts and adjusts the whole system more and more to the given problem.

Using reactive planning by the supervisor we were able to automatically solve most of the examples of [DF94] and [AD93] without any interference by the user: no team selection, no parameter adjustment by the user was necessary, just planning using general but vague knowledge was enough (see section 5). The knowledge-based reactive planning approach used by the supervisor of a team presented in this paper allows for a much easier extension of the system to new domains than the auto mode of Otter (see [Mc94])does. Here the code must be changed in order to deal with new domains. This auto mode was, like team work, designed to provide a fully automatic theorem prover to users that do not want to learn all about the tricks of a prover in order to work with it.

Our paper is organized as follows : After this introduction we will give a brief survey of equational deduction by completion which is the task we want to improve using team work. In section 3 we take a deeper look on the team work method and it's use for distributing automated theorem proving. In section 4 we concentrate on the tasks of the supervisor in a team and show how we represent the knowledge he needs and how he can use this knowledge to plan our distributed proof process. In section 5 we report on our experiences and discuss how the supervisor guides the search for a proof. In section 6 we relate our work to other works in the field of planning in AI. Finally in section 7, we give a conclusion.

2 Automated theorem proving and completion

Because this paper is mainly aimed to analyze and describe the planning aspect of team work done by the supervisor, we only give a very brief introduction into automated theorem proving, equational theorem proving and the completion method for equational proving. For more details we refer to [AD93], [CL73], [HR87] and [BDP89].

Theorem proving means solving the following problem :

Given: A set A of axioms and a theorem T to prove.
Question: Is T a logical consequence of A ?

In equational theorem proving A = {s_i = t_i— i=1,...,n} is a set of (all-quantified) equations and T is an equation u = v, too.

All successful methods for automated theorem proving, if equality is involved, are based on two kinds of inference rules : generation rules and contraction rules. The generation inference rules add new facts to the data base. These facts are derived either from the axioms alone (as in the case of equational theorem proving by completion, i.e. the critical pair generation) or from both, the axioms and the theorem T (as in the case of resolution and paramodulation). The contraction inference rules change or delete facts from the data base. A well known contraction inference is the reduction (or demodulation) that uses an equation l = r as rule $l \to r$ in order to exchange instances of l in a fact by the appropriate instance of r. Then a fact is in normal form, if no reductions with any equation of the data base is possible.

From a theoretical point of view we need for an automated theorem prover, besides the inference rules, fairness criteria for the use of the inference rules. These criteria guarantee that each application of an inference rule that is possible indeed will at some time be made. But for challenging examples there are many possible inferences and a systematic application results both in enormous run times and the need of much (memory) space. For such examples the prover very often has an agenda of 50,000, 100,000 or even more inference rule applications.

Therefore implementations of automated theorem provers use strategies and heuristics to select the next inference rule to apply and the facts these rules should work on. A strategy guarantees theoretical completeness whereas for heuristics it is possible that the prover will not find a proof even if there is one and enough time and space are provided. Many theorem provers allow the user to choose between various strategies and heuristics. But two problems remain. First, for a given problem there may be no appropriate strategy or heuristic implemented in the system. Second, even if there is a good one implemented, how does the user know which one is good ?

We tackled both problems with our team work method as we will demonstrate in the next section.

3 The Team work method

The team work method is inspired by project teams in business companies. Due to their temporary existence they allow an exact tuning to the problem they have to solve. This tuning is achieved by the supervisor of the team who chooses always the team he thinks is best suited for the current status of the solution of the given problem.

So one major task of the supervisor in a system based on the team work method is a planned selection of the team members. This planning must also allow a fast reaction to problems with the selected teams or when unexpected breakthroughs occur. We will discuss this task of the supervisor in more detail in the next section. In this section we will look at the other components of a team, the experts and referees, and their tasks and how they compete and cooperate to solve a given problem.

Experts are those components that work on the problem solution. They have to generate new facts and goals. Therefore for each expert there has to be a processor. If there are more experts than processors (which is usually the case) then the experts have to compete with each other in order to get a processor for the next working period. Each expert uses other methods to gain new data and represents a different view on the problem than his colleagues.

In automated theorem proving one achieves these distinctions by using different selection strategies and heuristics for the next inference step in the experts. There are many many criteria that can be used to get different heuristics. In our team work completion system we use, beside others, heuristics that

- focus on parts of the set of known facts,
- focus on statistical properties of the facts (for example the number of symbols),

- focus on special properties of the method used by the experts, i.e. completion,
- focus on the goal to prove.

Experts compete for processors, but a characteristic of a project team is also that the members of the team cooperate with each other. As a result of this cooperation, problems can be solved much faster than by a team member, i.e. an expert, alone. A good team is able to solve a problem faster than an expert alone even if we consider as time for finding the solution the sum of the times needed by all members (instead of time you can also be interested in the costs, but you will make the same observation). So cooperation leads to **synergetic** effects.

The team work method achieves cooperation between the experts by using referees and team meetings. After certain periods of time (the period length is determined by the supervisor) in which each expert works on the problem independently, a team meeting takes place. Before the meeting, each expert and his work so far is evaluated by a referee. The tasks of the referees are to compute a measure for the success of their experts and to select good results of the experts. The measure of an expert is used by the supervisor to determine the team members of the next round. The selected results are used to generate a new data base of facts and goals that is the starting point for the next round. Note that during a team meeting the main work is done by the supervisor. All processors, but the one running the supervisor, are most of the meeting idle.

The main problem in designing referees is how to compute measures for the overall behaviour of an expert on a problem and for the usefulness of generated results. For both measures there are statistical criteria that have proven to be quite successful in our application, automated theorem proving. Our different referees use a weighted sum of the following numbers (and some more, specific to the inference rule system used) to measure an expert (different referees are using different weights):

- the size of the data base,
- the number of contraction inferences that were performed, split into the number of contractions of facts and the number of contractions of goals,
- the number of potential inferences,
- the quotient of the average weight of the last few performed inferences and the average weight of all performed inferences.

The last number indicates whether the heuristic used by the expert is getting better at the moment (quotient smaller than 1) or getting worse (quotient larger than 1).

Note that during a proof attempt it is often necessary to change the referee of an expert. Whereas at the beginning of an attempt one expects a growing size of the data base and is therefore more interested in experts that have performed many contraction inferences, later in a proof one want to give such experts a good measure that can keep the size of their data bases stable or even manage to shrink it.

In order to select good results we use similar statistical criteria, but compute them for one fact, for example the number of contractions performed with this fact. Also we give the referees a maximal number of facts that can be selected and a minimal measure that each selected fact must at least achieve. So experts that only produce bad results can not add even some of their results to the new starting data base. By this we avoid blowing up the search space with unnecessary results.

Before we can concentrate on the planning aspects of the supervisor, we have to describe how the new starting data base is constructed. In order to guarantee completeness of our prover using the team work method we use the data base of the expert with the best overall measure as basis. Then we add the selected results of the other experts to this basis. This way, the whole progress of one expert (the best one !) is part of the new data base, but the good results of the other experts can improve this data base significantly thus leading much faster to a solution.

4 Selection of experts and referees by planning

As in the case of human project teams, systems based on the team work method rely on a good use of the given resources, i.e. the processors. The supervisor is responsible for the future assignment of these resources. In order to find such an assignment he has to use planning.

The main problem one has to face if one wants to use planning in theorem proving is a *general vagueness of all information* one can use. In contrast to areas like the blocks world or even movement planning for robots, there are no operators with fixed pre- and postconditions that can be put together in order to obtain an executable plan. In theorem proving the only candidates for such operators are the inference rules themselves and using them would lead back to the initial proof problem.

Instead one has to use planning on an abstract level that allows an easy search for a plan to a given problem. But such a plan may not be executable. By executable we mean that all preconditions a step of a plan needs are fulfilled when this step is done on the level of the inferences. A step may not be executable, because the outcome of prior steps delivered not the expected results. In general, one can say that the more knowledge about a problem is available the better a plan can be constructed and the more reliable this plan is during execution. The known planning approaches to theorem proving, for example [Bu88] or [SD93] (see section 6), require a lot of very concrete knowledge to be able to operate.

The abstraction level that is provided by the experts and referees of team work is much higher than the level used by the other approaches. This means that the information we have to a given problem is very vague. Therefore we have always to expect that a generated plan is wrong and we have to deal with this problem. Also one can not do a complete planning before doing any inference steps in the prover, because the more future steps one plans the more vague is the information about the outcome of these steps. A main reason for the vagueness of the used information is that, as already stated, in most cases one can not detect appropriate subproblems to a given proof problem a priori (the other approaches rely heavily on the detection of such subproblems).

Using team work we have to face a second problem when doing planning. The component responsible for planning is the supervisor, the component that represents the *bottleneck of our distributed system.* Therefore we have to do the planning in a certain, but varying, time limit that is small in contrast to the time of a working period. We hope to deliver better results with increasing planning time, but this can not be guaranteed (see later in this section).

Our solutions to these problems are first team work itself that allows in form of the experts to follow several promising directions at once, thus allowing some vagueness in the information used by the system. The referees enable the system to determine which of the directions is the most promising one and which of the other directions can provide good auxiliary results. As general planning approach we use ideas from *planning reactive behavior* (see [Mc90], [Be91]). These ideas, combining a long-term memory with knowledge about earlier proof attempts in a domain of interest and a short-term memory of the behaviour of experts on the actual problem, allow for adequate reactions to problems and unsuccessful plans and make replanning easier. The second problem is solved by implementing our planning approach as *anytime algorithm* (see [BD88]) that allows to stop with planning at any a priori given time.

The main concept that is used by the supervisor for planning is the **domain** of a given example. As in other proof planning systems, a domain is a collection of information about a set of proof examples one is (or was) interested in. In our system a domain is a collection of facts (defining the domain), consequences of these facts, methods useful for these facts and other information. The following example shows the domain description for the domain "ring".

domainname:	ring
signature:	+:2 , 0:0 , -:1 , *:2
equations:	x+0 = x ; x+(-x) = 0
	(x+y)+z = x+(y+z) ; x+y = y+x
	(x*y)*z = x*(y*z) ; (x*y)+(x*z) = x*(y+z)
	(x*y)+(z*y) = (x+z)*y
consequences:	0+x = x ; (-x)+x = 0
starting team:	ADD-WEIGHT ; ADD-RWEIGHT
middle team:	ADD-WEIGHT ; ADD-RWEIGHT
end team:	ADD-WEIGHT ; GOAL-SIM
superior domain:	group
specialized domains:	boolean ring
similar domains:	non-associative ring

Before we can go into detail how domains are used for planning we have to show how domains can be detected in order to be of any use. There are several possibilities how to determine weather a given example belongs to a domain or not. They differ in the amount of deduction used and therefore in the chances that a domain is detected.

If one wants to use as little deduction as possible then one should first try to find a match Σ from the signature of the domain (slot **signature**) to the signature of the example and then test for all facts of **equations** instantiated by Σ, if they are in the facts (i.e equations) that describe the example (up to renaming of variables). In practise this method has the problem that other equations of the example may reduce equations that are in **equations**. Then they can not be detected by the method although the example is in the domain.

The other extreme is to find a signature match Σ such that one can prove for all facts s=t in equations that $\Sigma(s) = \Sigma(t)$ is an equational consequence of the set E that defines the example. Instead to show one goal one would have to show many of them which is very difficult, because there may be domains to which an example does not belong (meaning that one has to prove that an equation is not a consequence of E which can lead to an infinite computation).

We have chosen a method between those two extremes. We check, if there is a signature match Σ such that for all instantiated elements of **equations** the normal forms of these equations are either identical or subsumed by an equations of the example. This overcomes the problems of the first method, but is also guaranteed to be decidable. Testing for domains is done in team work by a so-called "domain detection specialist" whose only task is to check many domain descriptions and to report those domains for which such a match can be found to the supervisor. In addition, this specialist can also report the equations of the slot **consequences**, instantiated by Σ, to the supervisor (or a referee's selection of these equations).

Using a domain detection specialist as member of the team of the first working period the supervisor can take advantage of the other information he has about the detected domains.

During the first team meeting the planning task of the supervisor is to select a plan skeleton. A plan skeleton is the central part of the description of a domain of the supervisor. It describes three good teams to use for problems of the domain. These teams should be used in the three different phases we have observed in many proofs. The starting phase is characterized by a growing number of facts (slot **starting team**). In the middle phase the number of facts both decreases and then increases, again (slot **middle team**). In the end phase, the proof is completed with a few further inferences (slot **end team**). We developed special experts for the end phase that use criteria that are based on similarities between the goal and the facts (see [DF94]). It is obvious that it is not easy to determine in which phase a proof attempt is. Especially the detection of the middle phase is very difficult. Currently we define the start of the middle phase as the team meeting in which most of the members of the starting team (that are not also members of the middle team) either are not member of the actual team anymore or have a measure that is below a predefined percentage of the best expert.

In the first team meeting the supervisor has to determine which of the detected domains and thus which of the plan skeletons he will use. It is very seldom that exactly one domain is detected. For example, if the domain "ring" is detected then also the domain "group" can be found, because it is part of the domain "ring". Even worse, a given proof problem can involve different domains that have no equations in common.

The supervisor grounds his decision mainly on the hierarchical information he has in the domain descriptions, in the slots **superior domain**, **specialized domains** and **similar domains**. Candidates for the domain the supervisor will concentrate on are all detected domains that are no superior domain to another detected domain. From these candidates the supervisor also eliminates those domains that are similar domains to a superior domain of another candidate. If we then have more than one domain as candidate left, we have no further knowledge available and therefore the supervisor chooses one random domain out of the candidates. The experts of the starting team of this domain will be members of the next working team.

If there are more processors available than there are experts in **starting team**, the supervisor will use also the starting teams of other detected domains. If the chosen domain has specialized domains, then a domain detection expert for these domains will also be a member of the team (if there are processors left, after the starting team of the detected domain is appointed to processors). If there are even more processors available, the supervisor will select experts that are known to cooperate well with already selected experts (see later). The supervisor stores all information he has received from referees and all decisions he has made in a file for future use (we have to use a file, because the processor that runs the supervisor can change during a proof attempt).

In every further team meeting the supervisor has to perform the following actions after he has received the reports of the referees :

1. Compute the time he has for selecting a new team.
2. Determine, whether the used plan skeleton is still good and whether the phase should be changed. If a change of the plan skeleton is necessary, a new skeleton and the appropriate phase has to be selected or the supervisor can decide to work without a domain (and a skeleton).
3. Choose the members of the next team.
4. Compute the length of the next working period.

The first task is easy, because we want the periods the supervisor is active to be very short. Therefore we do not allow the supervisor to use more time than one percent of the length of the last working period. There are exceptions dealing with very short working periods (less than 5 seconds).

We have given the supervisor two reasons when to look for further domains and therefore for a different plan skeleton. The first one is that a new domain is detected by a domain detection expert. If this detected domain is a specialization of the already used one, the supervisor immediately switches to the plan skeleton for this new domain. If the new domain is a superior domain to the already used one, then no change of plan is necessary. In all other cases a little more computation is needed to determine, whether a change should be made or not.

This computation is similar to the one made, when the second reason for domain change can be observed, namely that the experts chosen due to the plan skeleton are much worse than the best experts of the last team (according to the measures of their referees). For the decision which domain will be best one and therefore the further base of the planning process we rate the detected domains according to the following criterion:

> The supervisor sums up the measures of the experts of the teams of the plan skeletons of the domains. Each measure is weighted by 1 divided by the number of working periods since the measure was given. Therefore older information gets less credit. The domain with the best sum will be selected, if a certain threshold is reached. Otherwise the supervisor will use no plan skeleton and he will choose experts as we will describe for point 3 without use of domain information. If an expert was never member of a team during the proof attempt, he will get a measure of zero.

As the next step, the members of the team have to be selected. The best experts of the last working period and the members of the team of the current phase of the plan skeleton will be selected, with the exception that a member of the plan skeleton is not selected if his performance in the last working period has been much worse than the performance of the best expert. If there remain processors without an expert, the supervisor uses the following routine to find experts for them. If no expert of the plan skeleton was best expert, then it is possible that the proof attempt has reached another phase. Therefore the experts of the next phase of the skeleton get places in the team. If they perform well in the next working phase, the supervisor will assume that the next phase is reached.

Depending on the amount of time left for the meeting, weights for the members of the following groups of experts are computed (when they are not already in the team).

- the experts of the last team,
- the experts that work well together with the best expert of the last working period,
- the experts that work well together with the experts of the current phase of the plan skeleton,
- the experts that are suggested by other detected domains,
- all other experts.

In order to find the experts of these groups the supervisor needs information about experts. This information also includes data that is needed to compute the weight of the expert. Again we have chosen a frame representation for this information. The following is an example for such a frame.

expertname:	ADD-WEIGHT
robustness:	0.8
knowledge involved:	0.1
proof phase:	start: 0.6
	middle: 0.5
	end: 0.5
referees:	STATISTIC-1, start
	STATISTIC-4, middle
	STATISTIC-6, end
domains:	all
similar experts:	ADD-FWEIGHT-1 ; ADD-RWEIGHT
cooperative experts:	GOAL-SIM ; GOAL-MATCH
impossible experts:	none

The experts mentioned in **cooperative experts** are those we referred to as working well together with the expert described (in our example ADD-WEIGHT). All the other information is needed to determine a weight for this expert.

The computation of the weight of an expert according to the status of a proof attempt depends on the time the supervisor has left, again. The more time is available the more criteria are taken into account. These criteria are, in descending order :

- How good is the expert rated with respect to the detected domains and the phase of the proof ?
- How good were the results of the expert in earlier phases of this proof attempt ?
- How good does the expert cooperate with the already chosen team members ?
- How specialized is the expert ?
- How good is the robustness of the expert for the current proof phase ?

Each criterion will lead to a measure between -1 (bad) and 1 (good) and the weight of the expert is a weighted sum of these measures. If an expert that has never been member of a team has to be compared with experts that have been members, we use adjusted weights for the other criteria. Let us now take a closer look at the criteria.

The rating of the suitability of an expert for a domain takes into account, whether the expert is member of the team of the plan skeleton of the domain for the current phase and whether the domain is in the **domain** slot of the expert. If this is the case for all detected domains we would get a measure of 1. Note that experts that are not members of a plan skeleton of a domain may have the domain in their **domain** slot.

The measure that represents the history of the expert on the current proof attempt is computed as the mean value of the comparisons of the expert with the best experts of the working phases when the expert was member of the team. We get the comparison by dividing the result of the expert by the result of the best expert.

The measure for cooperation uses the slots **similar experts**, **cooperative experts** and **impossible experts**. For each already chosen expert we add 1, if the chosen expert is in **cooperative experts**, we add -1, if the chosen expert is in **impossible experts** and we add -0.1, if the chosen expert is similar to the expert we check at the moment. We add a small negative number, because this way the more similar experts we have in a team the more unlikely it would be to add another similar one. The sum is then divided by the number of chosen experts that are mentioned in the three slots.

If at least one domain was detected, we use the value of **knowledge involved** as indication for the specialization of the expert.

Finally, we multiply **robustness** with the appropriate value of **proof phase** to get a measure of the robustness of the expert.

If we have n free processors, the supervisor will choose the n experts with the highest measure. Note that the more time the supervisor has the more experts can be examined and the more knowledge can be used to come to a decision. But there is no guarantee that this decision will really improve over the time, because of the vagueness of the information we use.

Let us now take a look at point 4 of the actions the supervisor has to perform during a team meeting, the computation of the length of the next working period. If the plan has been successful so far, that means that the experts of the chosen plan skeleton or all experts of the last round have had good measures, then the lengths of the working periods increase linearly. When most of the experts did not have the time to perform more than 10 inference steps, the supervisor will use an exponential growing length.

If most of the experts of the next team are new, then it is difficult to tell whether the team will be good or bad. Therefore the length of the next working period will be shorter. How short depends on the number of facts that constitute the current problem description. The more facts the more time is needed to perform an inference step. In order to get useful measures from the referees, the experts have to perform several inference steps in the working period. If the team was successful, then it will get more time the next round, else other experts will be tried out.

5 Experiences

In the last section we described how the supervisor plans a proof attempt and reacts on the measures and results that he gets from the referees. In this section we will demonstrate that this planning enhances the performance of the whole system.

In [AD93] and [DF94] we showed by experiments that team work, without planning by the supervisor, can reduce the run time of a system on a proof problem dramatically compared to the run times of the used experts, when working alone. But these results have a drawback. We selected the team members of the teams. And these teams changed from example to example.

It is well known that all automated theorem provers have many parameters that can be adjusted to a given proof problem. Our important parameter was the team members. But this requires that the user of a theorem prover has much knowledge about the prover and its parameters, so that he can plan his proof attempts with the prover. But we want a system which can solve many, very different examples without the support of an user in an acceptable time.

We demonstrate that team work with planning enables us to build such a system by reporting results with examples from four different domains: propositional calculus (pl1 to 4), lattice ordered groups (lat1 to 17), boolean rings (bool5b) and rings (lusk6). For the descriptions of these examples see [DF94] and [AD93] or the appendix of this paper. In order to demonstrate what examples are in which domain we changed the names, but stated the original ones in parenthesis. We added the last two examples to make the detection of domains more difficult. The four domains provide a wide range of equational problems.

Table 1 documents our results. Besides the run times of a team using planning we also give the run times of the best team consisting of two experts we were able to find, the two experts that form this best team and the run time of the best expert working alone on the problem. We have chosen to use only two processors because with small resources a good use of them is very important. In order to allow a better comparison we restricted also the best user selected teams to two processors.

The team runs using planning were always started with the same starting team and the same system configuration. Besides the input of equations, a reduction ordering and the goal of an example and the start command no interaction between system and user took place. The alterations in the composition of the team were only effected by the actions described in section 4. The system configuration specially includes some basic data for computing the lengths of the working periods which play an important part in the run of a proof. Note that for the times of the best teams also the lengths of the several periods have been adapted to the specific examples.

The main observation in table 1 is that the team with planning needs a little more time than the best team for most of the examples but still can solve also those examples that no single expert can solve. It is clear that using planning we have to expect a certain overhead. Our analysis of the runs using our proof extraction and analysis tool (see [DS94]) showed that not the time for planning is responsible for the longer run times but the need to try out experts and the replanning that is involved when adjusting the team to a problem.

If we take a look at the experts that constitute the best teams it is quite obvious that even in one domain very different teams were needed. Especially in the domain propositional calculus for each example a different team was best. Therefore it can not be expected that the first plan to a domain always succeeds. Instead the reactive part of the system must detect bad experts and exchange them. And these bad experts in most cases do not contribute to finding a proof but slow the system down.

Interestingly, there are some examples (pl1, pl4, bool5b, lat7, lat8, lat10, lat11) for which the team with planning needs less time than the best team (which lets one ask why we call it the best team). While the run times for the lat examples are so short

example	team with planning	best team	best team members	best sequential expert
pl1	29.91	35.07	MaxWeight, CP-in-Goal	40.99
pl2	50.18	14.21	AddFWeight, GTWeight	45.02
pl3	202.89	72.19	Goal-in-CP, AddRWeight	297.16
pl4	84.86	96.47	Goal-in-CP, CP-in-Goal	—
bool5b	50.11	58.86	Goal-in-CP, AddWeight	—
lusk6	500.08	307.96	AddWeight, AddRWeight	3019.00
lat1 (mono1a)	0.35	0.05	Occnest, AddWeight	0.05
lat2 (mono1b)	0.36	0.05	Occnest, MaxWeight	0.05
lat3 (mono2a)	0.33	0.04	Occnest, AddWeight	0.03
lat4 (mono2b)	0.33	0.04	Occnest, AddWeight	0.03
lat5 (p1a)	0.71	0.28	Occnest, MaxWeight	0.27
lat6 (p1b)	0.68	0.47	Occnest, AddWeight	0.28
lat7 (p2a)	2.46	5.41	AddRWeight, Occnest	79.52
lat8 (p3a)	3.04	4.23	Occnest, AddWeight	4.14
lat9 (p3b)	2.94	2.62	Occnest, AddWeight	2.55
lat10 (p4a)	2.02	2.46	Occnest, AddWeight	1.84
lat11 (p4b)	1.93	2.06	Occnest, AddWeight	1.71
lat12 (p6a)	0.84	0.40	Occnest, MaxWeight	0.39
lat13 (p6b)	0.58	0.16	Occnest, MaxWeight	0.16
lat14 (p8b)	93.54	56.84	MaxRWeight, Goal-in-CP	—
lat15 (p9a)	22.58	8.66	Occnest, AddWeight	19.57
lat16 (p9b)	23.95	8.44	AddWeight, Occnest	50.95
lat17 (p10)	37.94	25.20	MaxRWeight, Goal-in-CP	—

Table 1: run-time comparison team with planning vs best team and best sequential expert (in seconds)

that this would not be significant, the other three examples proved to be interesting. Our analysis (and later experiments) showed that the better run times of the team with planning were due to the fact that experts that were not members of the best team -but chosen by the supervisor in the run using planning- provided results necessary for the proof a little earlier than the members of the best team. But they were not able to produce enough results to form with one of the members of the best team a better team (of two experts, which is why the term best team for the third and fourth row is correct). Using a team with three experts we were able to obtain a better run time than the team with planning.

So, in these examples planning allowed us a better use of the resources. But in general we have to expect that a change of the initial plan is necessary for many examples and that in some working periods some experts do not contribute to a proof.

If we compare the team using planning with the best sequential experts for an example we can observe that for small, easy examples the best sequential expert finds a proof faster. But for harder examples the team with planning clearly outperforms the best sequential expert and it still finds proofs when no sequential expert can. If we would compare our team using planning with a fixed sequential expert there would be much more examples for which this expert would find no proof. So the synergetic effect of team work can also be observed when the team uses planning.

Finally we have to point out that in section 4 two ways of using a detected domain has been described: first in order to add known consequences, second for planning purposes of the supervisor. In all the examples listed in table 1 a domain was only used in the second way, in order to emphasize the planning aspect of a domain. (It is obvious that adding suited results will extremely reduce the run time of the prover.)

6 Related Work

The work presented here is related to three areas of artificial intelligence, namely automated theorem proving, planning and distributed artificial intelligence (DAI). The first work that is related to the first two areas is due to A. Bundy (see [Bu88]), who invented the term **proof planning**. He concentrated on inductive theorem proving and used a STRIPS-like (see [FHN81]) planning approach. He invented so called tactics that are similar to the operators that can be defined in STRIPS. A proof attempt consists of two phases, a planning phase, where on a meta-level a proof is constructed using the tactics, and a proof phase, where the selected tactics are evaluated on the level of inference rules.

The problem of this approach is that the domains of the proof problems have to be understood very well, so that it will be possible to find appropriate subproblems to a proof problem. In equational or first-order theorem proving this is not the case as we stated in the introduction. The information about domains we have accessible is much too vague to allow the use of Bundy's approach.

In the area of planning we were inspired by the works of McDermott and Beetz on planning reactive behaviour (see [Mc90], [Be91]). Here planning was intended to help a robot navigate through an area and perform certain tasks with limited time available for planning. This limitation is also an important point of our approach. Now, robot control and automated theorem proving are very different areas, but, as stated before, both involve the necessity to deal with vague information. By use of several experts we have the possibility to choose the situation we want to continue on, which is not possible for only one robot.

In the areas of planning and distributed AI the research mainly concentrates on planning for autonomous agents, i.e. systems without a central control (see for example [DL87]), or on central planning of tasks that require coordination, because there are dependencies between the actions of the agents (for example plans for several robots, see [Ro82]). As the supervisor is the central control of the team and theorem proving using team work is a task where no dependencies occur, we have easy solutions to most of the problems addressed in these papers.

7 Conclusion and Future Work

We have presented a distributed theorem proving method where planning of the assignments of agents to the processors allows us to improve significantly the number of theorems that can be proved without the user fiddling with parameters of the prover. Although the run times of the version of our prover using planning are slower than the run times of the best known teams for the problems we were able to prove examples from different domains to which none of our sequential provers could find a solution thus still showing synergetic effects. Further, reactive planning enables us to prove examples from one domain where the best known teams differ from example to example.

Our approach to proof planning allows us to deal with knowledge about domains that is vague and can even be contradictory. This is due to the competition of the experts in the teams. So far, all other approaches to proof planning require exact and often total knowledge about a domain of interest in order to achieve satisfactory results. Furthermore, the addition of new domains to our system is easy because of the explicit representation of the knowledge about a domain by frames.

The detection of subproblems and the use of special methods to solve them that are a characteristic of other approaches can also be integrated in our approach without loosing the ability to deal with vague information. This is one direction in which we want to investigate in the future.

Other topics of future research are to automate the generation of domain information by learning from examples for the domain and the improvement of planning by not only selecting known experts but also by generating new experts using parameter adjustments of generic experts.

References

[AD93] **Avenhaus, J.; Denzinger, J.**: *Distributing equational theorem proving*, Proc. 5th RTA, Montreal, LNCS 690, 1993, pp. 62-76.

[BD88] **Boddy, M.; Dean, T.**: *An Analysis of Time-Dependent Planning*, Proc. 7. National Conf. on AI, Minneapolis, 1988, pp. 49-54.

[BDP89] **Bachmair, L.; Dershowitz, N.; Plaisted, D.A.**: *Completion without Failure*, Coll. on the Resolution of Equations in Algebraic Structures, Austin (1987), Academic Press, 1989.

[Be91] **Beetz, M.**: *Decision-theoretic Transformational Planning*, Internal report, Yale University, 1991.

[Bu88] **Bundy, A.**: *The use of explicit plans to guide inductive proofs*, Proc. 9th CADE, 1988.

[CL73] **Chang, C.L.; Lee, R.C.**: *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[De93] **Denzinger, J.**: *TEAMWORK : A method to design distributed knowledge based theorem provers* (in German), Ph.D. thesis, University of Kaiserslautern, 1993.

[DF94] **Denzinger, J.; Fuchs, M.**: *Goal oriented equational theorem proving using teamwork,* Proc. 18th KI-94, Saarbrücken, LNAI 861, 1994, pp. 343-354; also available as SEKI-Report SR-94-04, University of Kaiserslautern, 1994.

[DL87] **Durfee, E.H.; Lesser, V.R.**: *Using Partial Global Plans to Coordinate Distributed Problem Solvers*, Proc. IJCAI-87, 1987, pp.875-883.

[DS94] **Denzinger, J.; Schulz, S.**: *Recording, Analyzing and Presenting Distributed Deduction Processes*, Proc. PASCO'94, Linz, 1994, pp. 114-123.

[FHN81] **Fikes, R.E.; Hart, P.E.; Nilsson, N.J.**: *Learning and executing generalized robot plans*, in Webber, Nilsson (eds.) Readings in AI, 1981, pp.231-249.

[HR87] **Hsiang, J.; Rusinowitch, M.**: *On word problems in equational theories*, Proc. 14th ICALP, Karlsruhe, LNCS 267, 1987, pp. 54-71.

[LO85] **Lusk, E.L.; Overbeek, R.A.**: *Reasoning about Equality*, JAR 1, 1985, pp. 209-228.

[Mc94] **McCune, W.W.**: *OTTER 3.0 Reference manual and Guide,* Tech. Rep. ANL-94/6, Argonne National Laboratory, 1994.

[Mc90] **Mc Dermott, D.**: *Planning reactive behaviour: A progress report*, in J. Allen, J.Handler, A. Tate: Innovative Approaches to Planning, Scheduling and Control, Kaufmann, 1990, pp.450-458.

[Ro82] **Rosenschein, J.S.**: *Synchronization of Multi-Agent Plans*, Proc. AAAI-82, 1982, pp.115-119.

[SD93] **Sonntag, I.; Denzinger, J.**: *Extending automatic theorem proving by planning*, SEKI-Report SR-93-02, University of Kaiserslautern, 1993.

[Ta56] **Tarski, A.**: *Logic, Semantics, Metamathematics*, Oxford University Press, 1956.

Appendix: Input equations for the examples

The domain propositional calculus

These examples are based on a set of axioms for tautologies in propositional calculus given by Frege (see [Ta56]). All examples have the following set of defining equations and use an LPO with precedence C $\succ$ N $\succ$ T $\succ$ Ap $\succ$ Aq $\succ$ Ar as reduction ordering.

EQUATIONS C (T,x) = x
C (p,C (q,p)) = T
C (C (p,C (q,r)),C (C (p,q),C (p,r))) = T
C (C (p,C (q,r)),C (q,C (p,r))) = T
C (C (p,q), C (N (q),N (p))) = T
C (N (N (p)), p) = T
C (p, N (N (p))) = T

The task is to prove the following goals:

pl1 : C (C (Ap,Aq),C (C (Aq,Ar),C (Ap,Ar))) = T
pl2 : C (C (N (Ap),Ap),Ap) = T
pl3 : C (Ap, C (N (Ap), Aq)) = T
pl4 : C (C (N(Ap),N(Aq)),C (Aq,Ap)) = T

The example bool5b

This example states that in a Boolean Ring the associativity axioms are redundant. As reduction ordering we did not use the Knuth-Bendix ordering (with which this example is easy to prove), but an LPO with precedence n $\succ$ a $\succ$ o $\succ$ 1 $\succ$ 0 $\succ$ x0 $\succ$ x1 $\succ$ x2.

EQUATIONS o (x,y) = o (y,x)
a (x,y) = a (y,x)
a (x,o (y,z)) = o (a (x,y),a (x,z))
o (x,a (y,z)) = a (o (x,y),o (x,z))
o (x,0) = x
a (x,1) = x
a (x,n (x)) = 0
o (x,n (x)) = 1
CONCLUSION a(a(x0,x1),x2) = a(x0,a(x1,x2))

The example lusk6

This examples states that a ring where $x^3 = x$ holds is commutative. It is the most difficult example mentioned in [LO85]. Note that we do not use a special handling of the AC theory in our prover. The used ordering is an KBO with f:5 $\succ$ j:4 $\succ$ g:3 $\succ$ 0:1 $\succ$ b:1 $\succ$ a:1 (weights of the symbols are given behind the :-sign, weights of variables are 1).

EQUATIONS	j (0,x) = x
	j (x,0) = x
	j (g (x),x) = 0
	j (x,g (x)) = 0
	j (j (x,y),z) = j (x,j (y,z))
	j (x,y) = j(y,x)
	f (f (x,y),z) = f (x,f (y,z))
	f (x,j (y,z)) = j (f (x,y),f (x,z))
	f (j (x,y),z) = j (f (x,z),f (y,z))
	f (f(x,x),x) = x
CONCLUSION	f (a,b) = f (b,a)

The domain lattice ordered groups

All examples of this domain have the following set of defining equations and use an LPO with precedence i $\succ$ f $\succ$ n $\succ$ u $\succ$ 1 $\succ$ a $\succ$ b $\succ$ c as reduction ordering.

EQUATIONS	n (x,y) = n (y,x)
	u (x,y) = u (y,x)
	n (x,n (y,z)) = n (n (x,y),z)
	u (x,u (y,z)) = u (u (x,y),z)
	u (x,x) = x
	n (x,x) = x
	u (x,n (x,y)) = x
	n (x,u (x,y)) = x
	f (x,f (y,z)) = f (f (x,y),z)
	f (1,x) = x
	f (i (x),x) = 1
	f (x,u (y,z)) = u (f (x,y),f (x,z))
	f (x,n (y,z)) = n (f (x,y),f (x,z))
	f (u (y,z),x) = u (f (y,x),f (z,x))
	f (n (y,z),x) = n (f (y,x),f (z,x))

Since most of the goals to prove are conditional equations we listed in the following the additional equations and the conclusion we have to prove for all the examples of Table 1.

lat1	EQUATIONS	u (a,b) = b
	CONCLUSION	u (f (a,c),f (b,c)) = f (b,c)
lat2	EQUATIONS	n (a,b) = a
	CONCLUSION	n (f (a,c),f (b,c)) = f (a,c)
lat3	EQUATIONS	u (a,b) = b
	CONCLUSION	u (f (c,a),f (c,b)) = f (c,b)
lat4	EQUATIONS	n (a,b) = a
	CONCLUSION	n (f (c,a),f (c,b)) = f (c,a)
lat5	EQUATIONS	u (a,b) = b
	CONCLUSION	u (f (i (c),f (a,c)),f (i (c),f (b,c))) = f (i (c),f (b,c))
lat6	EQUATIONS	n (a,b) = a
	CONCLUSION	n (f (i (c),f (a,c)),f (i (c),f (b,c))) = f (i (c),f (a,c))
lat7	EQUATIONS	u (i (a),i (b)) = i (b)
	CONCLUSION	u (a,b) = a
lat8	EQUATIONS	u (a,b) = b
		u (c,d) = d
	CONCLUSION	u (f (a,c),f (b,d)) = f (b,d)
lat9	EQUATIONS	n (a,b) = a
		n (c,d) = c
	CONCLUSION	n (f (a,c),f (b,d)) = f (a,c)
lat10	EQUATIONS	u (1,a) = a
		u (1,b) = b
	CONCLUSION	u (1,f (a,b)) = f (a,b)
lat11	EQUATIONS	n (1,a) = 1
		n (1,b) = 1
	CONCLUSION	n (1,f (a,b)) = 1
lat12	EQUATIONS	u (1,b) = b
	CONCLUSION	u (1,f (i (a),f (b,a))) = f (i (a),f (b,a))
lat13	EQUATIONS	n (1,b) = 1
	CONCLUSION	n (1,f (i (a),f (b,a))) = 1
lat14	EQUATIONS	n (1,a) = 1
		n (1,b) = 1
		n (1,c) = 1
	CONCLUSION	n (n (a,f (b,c)),f (n (a,b),n (a,c))) = n (a,f (b,c))
lat15	EQUATIONS	u (1,a) = a
		u (1,b) = b
		u (1,c) = c
		n (a,b) = 1
	CONCLUSION	n (a,f (b,c)) = n (a,c)
lat16	EQUATIONS	n (1,a) = 1
		n (1,b) = 1
		n (1,c) = 1
		n (a,b) = 1
	CONCLUSION	n (a,f (b,c)) = n (a,c)
lat17	CONCLUSION	i (u (a,b)) = n (i (a),i (b))