Cooperation in Theorem Proving by Loosely Coupled Heuristics

Dirk Fuchs, Jörg Denzinger
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: {dfuchs|denzinge}@informatik.uni-kl.de

Abstract

We present a cooperation concept for automated theorem provers that is based on a periodical interchange of selected results between several incarnations of a prover. These incarnations differ from each other in the search heuristic they employ for guiding the search of the prover. Depending on the strengths' and weaknesses of these heuristics different knowledge and different communication structures are used for selecting the results to interchange.

Our concept is easy to implement and can easily be integrated into already existing theorem provers. Moreover, the resulting cooperation allows the distributed system to find proofs much faster than single heuristics working alone. We substantiate these claims by two case studies: experiments with the DiCoDe system that is based on the condensed detachment rule and experiments with the SPASS system, a prover for first order logic with equality based on the superposition calculus. Both case studies show the improvements by our cooperation concept.

1 Introduction

The most crucial step, with regard to performance, in a theorem prover that is based on generating new formulas (a so-called generating prover) is the selection of the next inference step typically out of an ever growing number of candidates. Even combining several inference steps into one to avoid intermediate results (which is the idea of hyper-resolution, for example) can slow down this growth only a little bit. Therefore, in a theorem prover a good management of a large amount of data and a good control of the extension process of this data is needed.

While indexing techniques (see [Gr96]) are well-suited in providing the good management of large numbers of formulas (clauses, equations, terms) several different research directions try to deal with the control of a theorem prover. Utilizing more restrictive inference rules can slow down the growth of the number of candidates but may require longer inference chains. Higher inference rates (by faster implementations or parallelization) may speed up finding proofs in many cases but they also speed up the growth of the number of possible inferences. Therefore, hard problems may remain out of reach even if a fast prover is utilized. An approach that has come into the focus of attention quite recently is learning control heuristics from previous proof experiences (see [Fu96], [DS96b]).

A problem that occurs within each of these research directions—especially in the area of learning—is the fact that very often a heuristic is capable of finding most of the steps necessary for proving a given problem rather quickly but there are a few steps the heuristic is not able to select out of the large set of possible steps (or at least not in an acceptable period of time). Usually it is even a problem to choose such a "not quite good enough" heuristic out of the large number of heuristics a theorem prover usually provides.

Cooperation of selection heuristics in form of different incarnations of a prover offers a solution to these problems. If several selection heuristics work in parallel the probability that a good heuristic is among them naturally is higher than the probability that one certain heuristic performs well. Furthermore, if these several incarnations interchange data, this data may contain information allowing one of the heuristics to conclude the proof. Obviously, the success of such an approach depends heavily on the amount of data to be interchanged (and when the interchange takes place). If many or even all new results of an incarnation are sent to the other ones the same negative effects as in case of higher inference rates occur (with additional cost for communication). If only a few results are interchanged, then the necessary results may not be among them.

In literature one can find two different ways for interchanging data between theorem provers: *demand driven* (as realized in the DARES system, see [CMM90]) and *success driven* (as realized in the TEAMWORK method, see [De95]). Demand driven cooperation means that a prover detects for itself that it misses a certain result and asks the other provers if they were able to deduce it. Success driven cooperation means that a prover has deduced a certain result that was very useful for it and therefore communicates this result to the other provers. Obviously, demand driven cooperation is not easy to achieve since it is fairly difficult to detect which results a prover needs to succeed (in fact, if we could solve this problem then in general a good control would be no problem).

In this paper we will examine success driven cooperation (with a slight twist into the direction of demands of other incarnations) by loosely coupled heuristics (the *CLCH approach*). Similarly to the TEAMWORK method we use *referees* to determine the results to be interchanged. This interchange takes place in regular intervals, but contrarily to TEAMWORK there is no central control of the system by a supervisor (hence we only have a loosely coupled network of provers/heuristics). Furthermore, the results of each heuristic remain on its computer and are only augmented by the selected results of the other computers. This is in contrast to TEAMWORK that replaces facts of different heuristics by the facts of the "best" heuristic. Thus the interaction scheme of CLCH is simpler and easier to implement (no broadcasting is necessary).

The referees play a vital role in the CLCH approach. Each very good, but difficult to generate fact that is not communicated to another incarnation may be the fact this incarnation needs to complete the proof. Each bad fact that is communicated to another incarnation makes the work of this incarnation harder and may gravely disturb its selection heuristic. Therefore the use of as much knowledge as possible in referees is very important in order to meet the demands of receiving incarnations as good as possible. In order to combine this with the necessity of a small amount of communication, the CLCH approach uses two kinds of referees. *Send-referees* judge facts by retrospective criteria and knowledge about the identity (i.e. the heuristic) of the other incarnations and *receive-referees* judge the facts selected by the send-referees with regard to the actual search state of its receiving incarnation.

We demonstrate the usefulness of our CLCH approach by means of two case studies. On the one hand we employ CLCH for coupling different incarnations of the CODE system (see [FF97]) which is based on the condensed detachment rule resulting in the new system DICODE. On the other hand we present experiments in coupling different incarnations of the superposition based prover SPASS (see [WGR96]). Since DICODE is also able to employ the teamwork method we can give an experimental comparison with this cooperation concept. The studies concerning SPASS show the minimal implementational requirements for our method and offer some information on the integration of our concept into existing, not self-implemented provers.

2 Basics of Automated Deduction

2.1 Fundamentals

The general problem in automated theorem proving is given as follows: Given a set of facts Ax (axioms), is a further fact λ_G (goal) a logical consequence of the axioms? A fact may be a clause, equation, or a general first or higher-order formula. The definition of "logical consequence" depends heavily on the concrete domain one is interested in.

Commonly, automated theorem provers utilize certain *calculi* for accomplishing the task mentioned above. *Analytic calculi* attempt to recursively break down and transform a goal into sub-goals that can finally be proven immediately with the axioms.

Generating calculi go the other way by continuously producing logic consequences from Ax until a fact covering the goal appears (but there are also some generating calculi that use the goal in inferences). We shall here concentrate on generating calculi.

Typically a generating calculus contains several inference rules which can be applied to a subset of the given facts (that constitute the initial search state). Expansion inference rules are able to generate new facts from known ones and add these facts to the search state. Contraction inference rules allow for the deletion of facts or replacing facts by other ones, thus contracting the fact base (see [De90]).

A common principle to solve proof problems algorithmically with a generating calculus is employed by most systems: Essentially, a theorem prover maintains either implicitly or explicitly a set $\mathcal{F}^P$ of so-called *potential* or *passive facts* from which it selects and removes one fact λ at a time. After the application of some contraction inference rules on λ, it is put into the set $\mathcal{F}^A$ of *activated facts*, or discarded if it was deleted by a contraction rule (*forward subsumption*). Activated facts are, unlike potential facts, allowed to produce new facts via the application of expanding inference rules. The inferred new facts are put into $\mathcal{F}^P$. We assume the expansion rules to be exhaustively applied on the elements of $\mathcal{F}^A$. Initially, $\mathcal{F}^A = \emptyset$ and $\mathcal{F}^P = Ax$ (or $Ax \cup \{\lambda_G\}$, as for example in resolution). The indeterministic selection or *activation step* is realized by heuristic means. To this end, a heuristic $\mathcal{H}$ associates a natural number $\mathcal{H}(\lambda) \in \mathbb{N}$ with each $\lambda \in \mathcal{F}^P$. Subsequently, that $\lambda \in \mathcal{F}^P$ with the smallest weight $\mathcal{H}(\lambda)$ is selected. In order to break ties between facts with the same heuristic weight it is possible to use another heuristic. Due to efficiency reasons ties are usually broken according to the FIFO-strategy ("*first in–first out*").

2.2 Condensed Detachment

A typical example for generating calculi is the inference system $\mathcal{CD}$ which contains the inference rule *condensed detachment* (`CondDet`) (see [Ta56] and [Lu70] for motivation and a theoretical background). Since $\mathcal{CD}$ contains only one expansion and one contraction inference rule it is very simple. But nevertheless resulting proof problems can be very challenging. Therefore, condensed detachment was chosen as a test domain by several researchers before ([Pe76], [MW92], [Sl93], [Wo95]) and the choice of condensed detachment as our test domain surely is justified. The rules of the inference system $\mathcal{CD}$ manipulate first-order terms. These terms are defined as usual, involving a finite set $\mathcal{F}$ of function symbols and an enumerable set of variables $\mathcal{V}$.

`CondDet` in its basic form is defined for a distinguished binary function symbol $f \in \mathcal{F}$. `CondDet` allows to deduce $\sigma(t)$ from two given facts $f(s,t)$ and s' if σ is the most general unifier from s and s'. $\mathcal{CD}$ contains—besides the expanding rule `CondDet`—the contracting rule `Subsum`. This rule allows for the deletion of a fact t if a fact s and a substitution σ exists such that $\sigma(s) \equiv t$. A proof problem $\mathcal{A} = (Ax, \lambda_G)$ is solved if a fact subsuming the goal can be deduced.

2.3 Superposition extended with Sorts

The theorem prover SPASS ([WGR96]) we chose to experiment with is an automatic prover for first order logic with equality. It is based on the superposition calculus (see [BG94]). The inference rules of the superposition calculus can be divided into expansion and contraction (also called reduction) rules as we have seen before. The expansion rules (ordered inference rules) contain the common rules of the superposition calculus, i.e. superposition left and right, factoring, equality resolution, and equality factoring. The reduction rules contain well-known rules like subsumption and rewriting. Furthermore, SPASS utilizes additional reduction rules, as the deletion of tautologies and the condensing rule, that allows to replace a clause C by $\sigma(C)$ if $\sigma(C) \subset C$. Since SPASS recognizes unary predicates as sorts ([We93]) the inference system of SPASS is extended by rules that apply to some special sort information. These rules have to be applied to facts before they can be involved in "normal" expansion and contraction rules. Because of the fact that the rules can be regarded as expansion rules we do not distinguish between ordered inference rules and rules that apply to some special sort information in the sequel.

3 The CLCH Approach

The general idea of the CLCH (Cooperation by Loosely Coupled Heuristics) approach is to interchange selected facts between several incarnations of a theorem prover (that use different heuristics for the activation step) in regular time intervals. The selection of the facts is the task of so-called referees. In the following we will give a more precise and detailed description of this general concept and we will examine the task of the referees in more detail. Finally, we will instantiate the concept for the case of the condensed detachment prover CoDe and the superposition based prover SPASS.

3.1 The general concept

The CLCH approach requires several incarnations of a generating theorem prover (that we will call agents in the following) running on different computing nodes and using different selection heuristics for the next step to do. All incarnations of the prover are simultaneously given the problem to solve, together with a schedule of cooperation phases in which they interchange facts (and each incarnation is assigned one or more referees, as described shortly).

Since we use incarnations of the same prover we can be sure that all of the agents in this distributed system use the same internal representations. Furthermore, as generating provers these agents are able to integrate new facts in their search state since this is required by the inference rules. The new facts are provided regularly during the cooperation phases by the other agents. So to say, the agents speak the same language and can make use of messages from the other agents.

Obviously, an agent should not communicate all new facts it generated since the last cooperation phase to all other provers. This would force a receiving agent to perform the processing of all these facts (i.e performing all possible inferences involving them). Then the cooperation of the provers would result in not much gain. In addition, communication is much more expensive than computation. Hence, a general goal of distributed systems is to limit the amount of information that has to be communicated. Instead, only a few selected facts should be communicated between two agents, namely such facts that promise to be useful for the receiving agent. The selection of these facts is the task of so-called *referees*.

During a proof attempt the provers repeat the following cycle: In the *working phase* each prover applies the inference rules of the calculus, controlled by its heuristic, to the data base of facts it started with. After a fixed period of time a *cooperation phase* takes place: Referees determine for each prover facts that should be integrated into the data bases of the receiving provers. These facts are integrated and then a new cycle (utilizing this new data base) starts. In the sequel, we shall discuss at first several possibilities of utilizing different kinds of knowledge for selecting facts. Moreover, we introduce an architecture that allows us to employ the different kinds of knowledge resulting in a system that has both success driven and demand driven features. Eventually, we sketch some implementational aspects.

3.1.1 Utilizing different kinds of knowledge for selecting facts

Both, behavior of a referee and architecture of our distributed system depend heavily on the knowledge that the referees utilize for selecting outstanding facts. It is clear that a referee responsible for the selection of outstanding facts of a prover (sender) must have at least *local knowledge*, i.e. it must know the system of facts to choose from. It is unclear, however, if and how knowledge about other provers, i.e. possible receivers of facts, could be efficiently used. In general there is a wide spectrum of knowledge about the receivers of facts that ranges from local knowledge (only information on the sender, no information on the receiver) to *global knowledge* (total information on both, sender and receiver). Utilizing different kinds of knowledge in a referee entails different behaviors of the referee and even different architectures of our distributed system as we will see in the sequel. We discuss at first the two extremes, local and global knowledge. Then we will show which changes of the behavior of a referee and the architecture of the system are necessary if we start from local knowledge and employ more and more knowledge about the receivers of facts. We distinguish between three different concepts to enrich local knowledge: Employing local knowledge and knowledge about the heuristic of the receiver, local knowledge and knowledge about the current needs of the receiver, and finally a combination of local knowledge and knowledge about both, heuristic and current needs of the receiver.

Global knowledge means that a referee has, beyond the information on the system of facts to choose from, also complete information on the receiver of facts at its disposal. This information contains, e.g., information on the search strategy (heuristic), on the search state (the current data base of facts) of each receiver and even information on earlier search states (i.e. the history) of each incarnation of the prover. By utilizing global knowledge, a referee can perform an optimal selection w.r.t. a sender S and a receiver R. This is due to the fact that the referee is able to select such facts from S that seem to be most profitable for R considering its system of facts and its search strategy. There are some practical deliberations, however, which make it very difficult or even impossible to employ global knowledge for selecting facts. Since the referee would need information on the system of the sender as well as on the receiver to realize a selection as sketched before, a tremendous communication overhead is unavoidable. In practice, global knowledge can only be utilized if sender and receiver work on a common blackboard memory, e.g. on a multiprocessor machine. Since we are interested in developing cooperating heuristics working on different computers, e.g. in a network of computers, employing global knowledge is impossible due to the high communication amount.

The other extreme is to select facts without knowledge about the receivers (using only local knowledge). Thus, criteria have to be developed to decide—without any hints on the prover that receives it—whether a fact is useful in general. In [DF96] such kinds of referees that mainly use retrospective views on the results to choose from are introduced. Considering the results mentioned in [DF96] it is possible to gain efficiency with this technique in spite of the fact that only vague criteria are employed. Moreover, a main advantage is that only one selection of facts of a sender is necessary to determine the facts that should be sent to all other provers. Therefore, for each prover only one referee has to be employed and hence the overhead caused by the cooperation phases is rather small. Furthermore, the amount of time for the selection does not depend on the number of agents in the cooperating system. Thus, increasing the number of cooperating agents does not decrease the efficiency. As we will see later, however, this technique is sometimes too primitive. Especially if the receiving provers need only a few facts to conclude the proof, an individual selection for each receiver is sensible.

The easiest method to extend local knowledge is to employ at least knowledge about the search strategy of each receiving prover. In the area of generating provers that means its heuristic. In [DF96] a referee is introduced which selects facts for a receiving prover considering its heuristic. As we will see later, the knowledge about the heuristic of the receiver enables a referee to estimate which consequences the integration of a fact into the system of the receiver might have. Thus, the quality of facts could sometimes be better estimated in comparison to the restriction of local knowledge. But note that employing knowledge about the heuristic of the receiver requires $n-1$ selection processes (if we have n provers), whereas only one selection was necessary before. Hence, we need $n-1$ referees for each prover and the number of selections of each agent increases linearly with the number of receiving agents.

The second possibility to extend local knowledge is to utilize knowledge about the current needs of each receiver thus adding a demand driven touch to the system. If a referee is able to utilize such kind of knowledge its task of selecting facts could be strongly simplified. But as we have already discussed the problem occurs that the referee must on the one hand know the system of the sender to select facts from it. On the other hand he must know the system of the receiver in order to recognize the current needs of it. Because of the fact that it is impossible to evaluate the complete systems of sender and receiver at a central point (due to the communication overhead) the following change of the architecture might be sensible to overcome the problem: At first, a *send-referee* could select some facts that seem to be important in general. (Note that local knowledge about the quality of facts is always available.) Thus, the probability is rather high that important facts are among them. After the transmission of this usually quite small set of facts (in comparison to a complete system of facts of a prover) to the receiver, the work of an individual *receive-referee* starts. This referee selects from the facts chosen from the send-referee those that seem to be useful w.r.t. the receiving prover's system. Thus, only one receive- and one send-referee is needed for each agent, i.e. the number of referees of each prover does not depend linearly from the number of agents. Moreover, the integration of knowledge about the individual needs of a prover is not paid for by a high amount of communication. This is due to the fact that not a complete system of facts is send to the receivers but only some facts selected from the send-referee. Nevertheless, the costs for selecting facts are higher in comparison to the costs necessary when employing local knowledge: We need two selection processes whereas we only needed one before.

Finally, it is possible to combine the two methods described previously and to employ beyond local knowledge, both, knowledge about the heuristic and the current needs of a receiver. In analogy to before we have to split the selection process into two ones and employ send-referees as well as receive-referees. Each send-referee is responsible to select some facts that are to be transmitted to a receive-referee. Since we want to employ knowledge about the heuristic of the receiver, we need $n-1$ send-referees for each prover. The output of send-referee i is then input of the receive-referee of prover i. The set of facts selected by each send-referee contains a common subset of facts selected due to local knowledge extended with facts selected individually for each receiver. The receive-referee of each prover selects—as we have seen before—from the facts received from all send-referees such facts that seem to be useful w.r.t. the current system of facts of the receiver. In order to accomplish its task it employs knowledge about the current needs of the receiver. This kind of selection entails the highest costs: we have to employ $n-1$ send-referees for each prover, i.e. the number of selections depends on the number of provers. Moreover, we need two selections until facts can be integrated into the systems of the receivers. But note that by utilizing this technique very much knowledge about the receiving provers can be incorporated into the selection process, without much amount of communication. So, this is also a combination of success driven and demand driven cooperation.

3.1.2 System architecture

Figure 1 depicts the architecture of a system employing CLCH (for the case of three cooperating provers). The figure presents the most general architecture, i.e. the architecture where the most knowledge can be integrated in. As one can recognize we have at each sender individual send-referees for each receiver. These referees select on the one hand for each receiver facts w.r.t. its heuristic. On the other hand they can send common facts selected according to local knowledge to the receiving provers. Each prover is associated with a receive-referee which filters facts from the data received from the send-referees of the other provers.

The dotted lines in figure 1 suggest that the architecture can be simplified if the referees fall back on less knowledge: On the one hand it is possible to constrict different send-referees horizontally (from the point of view of one prover). Thus, we employ only one send-referee instead of $n-1$ ones (in the case of n provers) and can only select facts according to local knowledge. In such a case the output of the send-referee is sent to all other provers in the system. A vertical constriction is possible, too. (Constriction of receive- and send-referee.) As we have discussed before, however, it is only possible to constrict different referees to one that is working at the sender site: Due to the high communication amount it is unwise to send the whole system of facts of the sender to

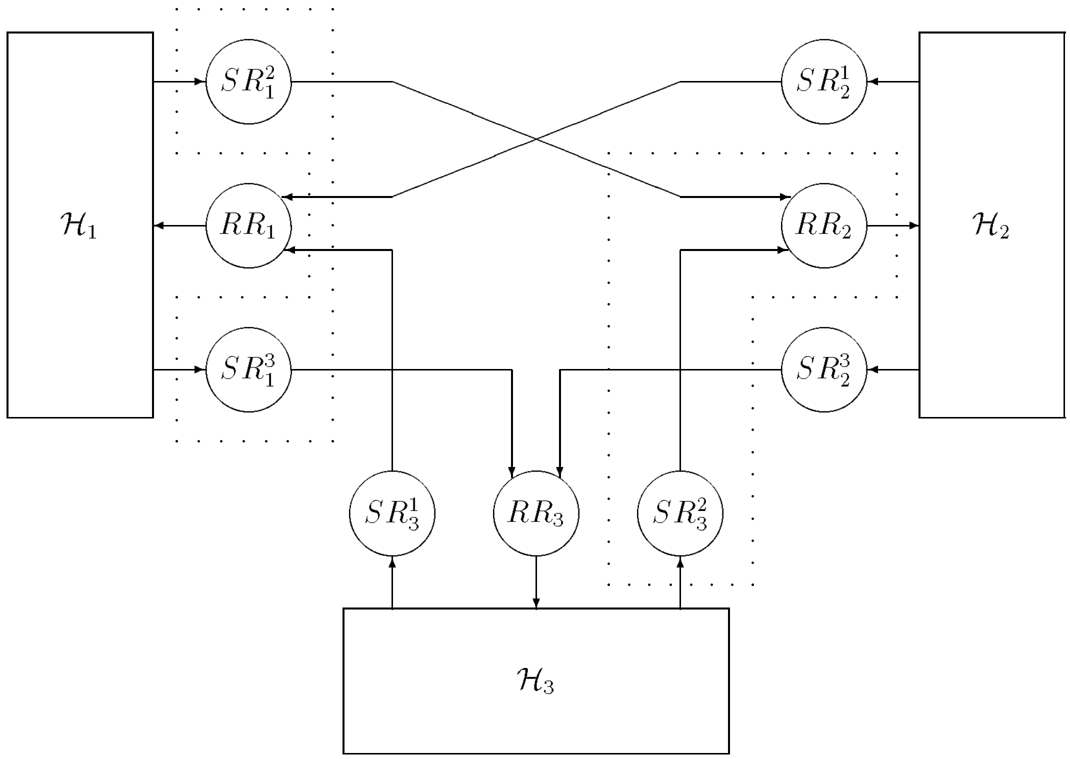

Figure 1: Coupling three heuristics with the CLCH-approach

the receive-referee. Furthermore, the selection functions of receive-referees are quite time-consuming because of the fact that they consider the needs of their associated provers (see below). Therefore, receive-referees should not be applied to a large set of facts. If we want to perform a vertical constriction we must hence resign the receive-referees.

Since different architecture models exist the question arises which architecture is the most suitable in a certain situation. Obviously, the question cannot be answered in general. Nevertheless, we want to give a few hints on this issue: If very good heuristics work in the network of cooperating provers—e.g. heuristics which fall back on learned knowledge—it is very probable that they already generate a lot of facts that contribute to a proof and need only a few additional facts to conclude it. This way, the referees can select from a set of facts that contains many facts well-suited for the problem. Hence, in such a situation it might be the right way to take the heuristic of the receiver into account and to select such facts that the receiver will not activate by itself due to its search strategy. Because it is sensible to employ only local knowledge and knowledge about the strategy of the receiver, we can resign the receive-referees and hence select facts more efficiently.

It is often the case, however, that only heuristics cooperate with each other that employ very simple syntactic criteria. In such a situation each prover usually generates a lot of facts that do not contribute to a proof. Hence, we have to choose from a set of facts that contains a lot of unnecessary facts. Thus, it is reasonable to employ as much knowledge as possible in the selection process, even if we have to spend more time for the selection of good facts. Therefore we must employ both, send- and receive-referees. Send-referees are necessary in order to employ local knowledge and knowledge about the heuristic of the receivers. The use of receive-referees is sensible because they consider the current needs of their associated provers. All in all we can say that the quality of heuristics has to decide which kinds of knowledge we have to integrate into the referees.

3.1.3 Implementational aspects

In order to implement a system based on the CLCH approach two design decisions have to be made, namely how to couple referees and provers and how to organize communication between the provers. For both decisions there are two alternatives.

Referees can be integrated into the prover thus creating an agent with different roles or they can be realized as external referees which means as separate processes. Applying the role concept, prover and referee are not separated physically but both are part of the same process and operate on shared memory (internal referee). Since the process is only allowed to play one role at a time access conflicts cannot occur. Such an approach facilitates the development of referees because it is possible to have access to internal data of the prover. If the selection process is split into two different ones, i.e. we use both receive-referee and send-referee for each agent, at least the receive-referee must be implemented as an internal one: This referee has to access internal data of the prover because it selects facts w.r.t. the current system of facts.

External referees are of interest if the CLCH approach should be integrated into already existing provers without much implementation effort. Realizing the referees as separate processes, possibly using the standard output of the prover as input, does not require many changes within the prover (which would be the case when choosing the other alternative). But note that this limits the amount of internal data of the prover the referees have access to. Therefore, external referees should only be employed if we implement send-referees. Since we have developed theorem prover and referees simultaneously or integrated program code into an available prover we have chosen the first alternative.

There are two different realizations of communication, namely asynchronous and synchronous communication. When employing asynchronous communication each process (or prover in our case) is allowed to initiate communication at any time by interrupting the receiving process. Synchronous communication is only possible if all communicating processes have reached an internal status in which they are willing to communicate. Since the CLCH approach favors communication at fixed moments (i.e. during the cooperation phases) one should choose synchronous communication.

3.2 Criteria for Referees

The quality of the referees influences heavily whether cooperation between the provers really takes place or not. Bad referees, i.e. referees that select only facts that will not contribute to a proof of a given problem, are only an additional obstacle for the provers that receive their selected facts. This is due to the fact that all the received facts have to be integrated into the data base of a prover (which requires some computational effort) and then these facts can cause additional efforts in many further computations. Additionally, unnecessary facts produce more unnecessary facts.

On the other side good referees will select at least some facts that contribute to a proof of a given problem. If these facts are not already in the data base of a receiving prover then the search process of this prover may lead to a success faster. Especially, if received facts would be generated very late by the heuristic of a receiving prover, then a substantial speed-up of the search is achieved.

All kinds of referees employ a *selection function* φ for the selection of facts. φ can employ several judgement functions $\psi_1, \ldots, \psi_n$ in order to select facts. These functions ψ_i associate a natural number with each fact λ which is considered in such a way that a fact is the better the higher the value $\psi_i(\lambda)$ is. φ selects eventually the facts with the best judgement. There are several possibilities how the judgements of n judgement functions $\psi_1, \ldots, \psi_n$ can be combined. It is possible, e.g., to construe one judgement function ψ employing the functions $\psi_1, \ldots, \psi_n$. In our experiments, however, we used either only one of the functions or selected via each function ψ_i a certain percentage p_i of facts.

In the sequel we present different concepts of referees. The main topic that we investigate is how to develop judgement functions, i.e. how to measure the quality of facts deduced from a prover. To this end, we demonstrate different ways in order to define the term "quality" if different kinds of knowledge are utilized. In order to describe different concepts of referees we distinguish between send- and receive-referees: Send-referees employ either local knowledge or knowledge about the heuristic of the receiver (its identity) and select facts from the system of the sender immediately. These facts are either filtered again (by receive-referees) or directly integrated into the systems of the receiving provers. Receive-referees use knowledge about the current needs of their associated prover that uses the current search state of the prover. The input of a receive-referee are facts filtered by send-referees of all sending provers.

3.2.1 Send-Referees

As we have discussed before each sender is assigned either one send-referee whose output is send to all other provers (if we employ local knowledge) or individual referees for each receiving prover (if we employ additionally knowledge about the receiver's heuristic). Independently from the knowledge integrated into the send-referee its selection follows certain fixed principles: A send-referee in a system based on the CLCH 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 will get those facts in the cooperation phases that pass through the filter and that are among the m selected facts (so, m is a parameter limiting the amount of facts that can be passed between the provers).

The filter predicate S, that is typically a conjunction of several conditions, is used to limit the set of facts that are eligible for transmission to other provers. Only facts λ with $S(\lambda) = \mathit{true}$ pass through this filter (so, no numerical evaluation is involved).

Typically, facts are filtered out that are (thought of as) redundant (with respect to the receiving provers). Redundant facts are all axioms and all facts that were selected in previous cooperation phases. But S can also be used to reduce the set of facts φ can select from even more, e.g. by filtering out all facts that were not generated since the last cooperation phase. Computing $S(\lambda)$ for a fact λ should be easy and fast in order to allow more complex (and time consuming) computations for φ.

The selection function φ is used to choose among the facts that passed S. As we have mentioned before, φ can employ several judgement functions $\psi_1, \ldots, \psi_n$ in order to select facts. The different kinds of knowledge a send-referee utilizes are determined by the judgement functions. If we have only judgement functions that fall back on local knowledge also the selection is only performed due to local knowledge. Hence we need only one send-referee. If only judgement functions are available that employ knowledge about the heuristic of the receiver we need $n-1$ send-referees (if we have n provers) and each of the referees uses individual judgement functions. Usually we will have both, judgement functions employing local knowledge and judgement functions using knowledge about the receiving prover's heuristic. In such a case we need $n-1$ send-referees, too, but have to perform the judgement according to local knowledge only once for all referees.

Finally, we have to deal with the question how to realize judgement functions, i.e. how to measure the quality of facts. We distinguish between functions founding on different kinds of knowledge:

If we select facts according to local knowledge the principle consideration is to define the quality of a fact according to the *success* of this fact recorded by the prover that generated it. Then, the quality of a fact is determined by its history during the proof attempt so far. This means that all inferences the fact was part of should be considered when developing a measure for the quality of a fact. Note that by means of such a retrospective view on the performed inferences it is possible to use a posteriori knowledge whereas typical search-guiding heuristics are only able to utilize a priori knowledge. Nevertheless, often some syntactical properties determine the quality a fact has had during the deduction process (see below).

We have developed two types of judgement functions considering this definition of quality, namely functions ψ_S and ψ_G. Judgement functions of the type ψ_S (statistical referee) use only retrospective criteria for measuring a fact, namely a weighted sum of the numbers of inferences a fact was involved in. The general idea is that facts that are good in one prover (according to this statistical data) will also be good in other provers and therefore should be transmitted to them. ψ_S counts the performed inferences during the inference process.

Functions of type ψ_G, however, try to estimate whether a fact will behave well or badly with the help of syntactical criteria. More exactly, they try to estimate how often a fact will be involved in "good" inferences, like subsumption of other facts, and "bad" inferences. Therefore, they measure mainly the generality of a fact because general facts are possibly often involved in good inferences and seldom involved in bad inferences.

If we fall back on knowledge about the heuristic of the receiver "quality" has to be defined in another way: From the point of view of a prover that receives facts the quality of a fact is determined by the effects this fact will have on the future search process. Therefore, it is at first important that the receiver has not activated the fact itself which would render the information useless. Moreover, if descendents of this fact are never activated and if also no other facts are subsumed by it, this fact is also definitely useless. Therefore, it is sensible to choose facts whose descendants are possibly activated by the receiver.

The knowledge about the heuristic of the receiver can be helpful in order to estimate this. We have developed the type of functions ψ_H ("heuristic") considering this definition of quality. The idea of ψ_H is to use the selection heuristic of the receiving prover (that also computes a measure of a fact) as main part of the judgement functions. Facts that are preferred by the receiving prover will also be preferred by the judgement function. If the heuristics of the receiving prover and the prover generating the facts differ very much the receiving prover has often not already generated the selected facts on its own.

But in general functions of type ψ_H have the problem that many selected facts were already generated by the receiving prover. One way to overcome this problem is a retrospective view, again, by looking at the ancestors of a fact. If the fact is fairly well-suited with respect to the heuristic of the receiving prover but there are ancestors of it that are not, then it is not very likely that the fact has already been generated by the receiving prover. Consider for example the fact λ that is inferred by the sender using a certain inference chain $\mathcal{I}$. Let the facts involved in this chain be $\lambda_1, \ldots, \lambda_n \equiv \lambda$. Since the sender was able to perform this inference chain we can assume that all of these facts have a small weight according to the sender heuristic. If λ has also a small weight w.r.t. the heuristic of the receiver ($\mathcal{H}_R$) it is possibly well-suited for it. If the heuristics of the sender and the receiver differ from each other the following phenomenon can occur: If there are facts λ_i, $1 \leq i < n$, that have a high weight according to $\mathcal{H}_R$, these facts are not preferred by $\mathcal{H}_R$ and the receiver is hence not able to perform the inference chain by itself. Therefore it is possible that the fact λ is really new for it, although λ has a small weight according to $\mathcal{H}_R$.

It is to be emphasized that such a criterion can only be employed by a send-referee that has access to internal data of a sending prover. It is impossible for a referee that is working at the receiver site to select facts with this criterion: In order to employ it, it would be necessary to know not only the facts to choose from but also the inference chains the facts were derived with. But sending of the information on both, facts and inference chains, is practicably impossible due to the enormous amount of communication involved.

3.2.2 Receive-Referees

Knowledge about the possible needs of the receiving provers can be integrated into receive-referees. They select from the facts they receive from the send-referees of the other provers some facts that are then immediately integrated into the system of their associated prover. The main advantage of receive-referees in comparison to send-referees is that they have access to internal data of the receiver, more exactly to the system of active and passive facts. Furthermore, they must only select facts from the set of facts they have received from the send-referees of other provers. Thus, they have to choose from a small set of facts and can therefore employ more sophisticated (and time-consuming) criteria than a send-referee. Nevertheless, the knowledge about the system of the receiver is somewhat limited and can only facilitate the selection a little bit. (In order to go beyond the limitation for each fact the whole future proof and the role of the fact in it has to be computed, which is not feasible.) Since the receive-referee is only able to estimate which consequences the integration of certain facts will have we must employ heuristic criteria again. Therefore, even the selection of a receive-referee might be insufficient in some cases.

In general there are two main principles how receive-referees could be designed leading to two kinds of judgement functions. We will only sketch their functionality here and describe them in more detail in section 3.4.2. Functions belonging to the first class ψ_{SG} (subgoal referee) judge facts with respect to their ability to contribute to the solution of certain *(sub-)goals*. Functions belonging to the second class ψ_U (usefulness referee) measure the *usefulness* of facts w.r.t. the inference process starting from the current system of facts of the receiver. The first principle means that a receive-referee should select facts that are in such a way similar to certain (sub-)goals that they are necessary to solve them. Since it is not the task of a receive-referee to perform inferences by itself simple criteria have to be found, how it can be estimated a priori if a fact contributes to the solution of a certain (sub-)goal. Facts which are judged with a positive value according to the second class of judgement functions should have positive influences on the inference process starting from the current system of facts. Such positive influences could be, e.g., that such facts are in future often used for subsuming other ones.

In order to select facts functions of both kinds could be used to construe one judgement function. The receive-referee could then employ this function for the selection of facts. In our experiments, however, we let one function from each kind select a certain number of facts. The result of the receive-referee then was the union of both sets.

In the following section we will compare the CLCH approach with the TEAMWORK method that is also a success driven cooperation concept. Then we will instantiate our CLCH approach in two different areas: On the one hand we instantiate CLCH for the case of a condensed detachment prover. The resulting prover DICODE can also work in a TEAMWORK modus and allows us to compare the two concepts empirically. In order to show that existing provers can easily be coupled with the CLCH approach we employ it on the other hand to couple different incarnations of the superposition based theorem prover SPASS, that was neither developed nor implemented in our group.

3.3 The TEAMWORK method

There are several similarities between the CLCH approach and the TEAMWORK method ([De95]), but also several differences that result in a different behavior of systems based on these cooperation (and distribution) concepts. In the following we will give a short introduction to the TEAMWORK method guided by the similarities and differences to the CLCH approach.

The TEAMWORK method is also a multi-agent approach but in addition to the two types of agents of the CLCH approach (provers and referees) there are two more types, namely a supervisor and specialists. Provers (that are called *experts* in TEAMWORK) and *specialists* work independently from each other for certain periods of time, each agent on its own computing node (similar to the CLCH approach). Experts use the same calculus but different selection heuristics while specialists may employ any (correct) means to generate new facts and can perform administrative tasks for the supervisor (which is the central control of a TEAMWORK-based system), too. One supervisor and several experts and specialists, each of them associated with a referee, form a team.

Similarly to the CLCH approach there is a cooperation phase at the end of each working phase that is called a *team meeting*. At the beginning of a team meeting each expert and specialist is judged by a referee that computes a measure of success indicating how good the work of its expert/specialist during the last working phase was and that selects outstanding facts. Only the second task is similar to the tasks of a referee in the CLCH approach.

In the second part of a team meeting the supervisor is active. It controls and plans the whole search process of the team of agents. It generates out of the data base of the expert with the best measure of success and the selected facts of the other experts/specialists a new start data base that is broadcast to all other computing nodes. The supervisor also selects the members of the team of the next working phase using a long-term memory about agents and domains of interest, with plan skeletons for good teams, and a short-term memory with data about the actual proof attempt. The detection of the domain of an example is done by specialists (see [DK96]). By exchanging experts/specialists with bad performance for other ones the supervisor is able to adapt the whole team to the given proof problem. Finally, the supervisor determines the length of the next working phase.

The main differences between the TEAMWORK method and the CLCH approach are as follows: TEAMWORK employs a flexible but centralized control by the supervisor whereas in the CLCH approach fixed teams without central control are used. Furthermore, the referee tasks differ: In TEAMWORK they are oriented on the "survival of the fittest" principle whereas in CLCH the judgement orients on the individual receivers (adding demand driven features) and thus is very flexible.

The centralized control with the competition of the experts and the adaptation of the team to the given problem is an advantage of TEAMWORK (over the CLCH approach). Since all agents start each working phase with the same (hopefully most advanced) data base it is possible in a TEAMWORK-based system to use very specialized selection heuristics that usually cannot find a proof in a bottom up manner but can lead to a proof very fast if they can start with an appropriate data base provided by other experts (see [DS96a] for an analysis of the synergetic effects of TEAMWORK). Due to the referees the useless results of these heuristics are *forgotten*. But in spite of these advantages there still are some disadvantages resulting from complex implementations that require a secure broadcast, the realization of the components as roles of an agent, and many additional configuration and data files for the planning process. In contrast, the CLCH approach is easy to implement and does not require many changes of an already existing generating theorem prover. Additionally, a system using the CLCH approach is guaranteed to be able to withstand the loss of a computing node (by using timeouts). Contrarily, a TEAMWORK-based system that loses the computing node of the supervisor has to be terminated.

Although the referee idea originates from TEAMWORK it is in a TEAMWORK-based system somewhat limited since the selected facts are integrated into one data base (that of the best expert). This requires either the use of criteria that are mostly retrospective or the communication structure has to be made even more complex in order to inform all referees of the heuristic of the winner (see [DF96]). The needs of other non-winning team members are not considered by the referees. The referees in the CLCH approach, as we have seen, do not have this limitation.

Since in TEAMWORK only the complete data base of the best expert survives the timing of the team meetings is fairly important. If there is not enough evidence that a certain heuristic performs well and will lead to a proof another heuristic may become the winner. But this means that the not-winning heuristic has to repeat its (possibly) important steps and improve on them until it becomes the winner in a later meeting. Therefore, it is possible that a proof is unnecessarily delayed. Although the adaptation process takes care of this general problem it does not exist at all in the CLCH approach. If the CLCH approach can be used on a prover that has some very good heuristics that only need a few additional facts generated by other heuristics to prove many examples (which can be provided using learning heuristics, see [Fu96] or [DS96b]), then the CLCH approach is sufficient to provide the necessary cooperation.

3.4 The CLCH approach for Condensed Detachment: The DiCoDe System

The DiCoDe system is based on a re-implementation of the CoDe system (see [Fu96]) that has had as main goal the improvement of the basic inference machine by using indexing techniques. Additionally, DiCoDe features two distribution concepts, namely the CLCH approach and the TEAMWORK method. In the sequel, we will describe the different control heuristics that can be used in DiCoDe, instantiate the referee concepts of section 3.1, and concentrate briefly on the completeness of DiCoDe when using the CLCH approach as distribution concept.

3.4.1 The Control Heuristics

DiCoDe features all the heuristics of the CoDe system and some additional ones that were developed considering the two cooperation concepts. The basic heuristic of DiCoDe—which is called LevW—computes the weight of a fact λ as the weighted sum of λ's level and term weight. The level $\delta(\lambda)$ of λ is 0 if λ is an axiom. Otherwise, $\delta(\lambda) = \max(\{\delta(\lambda_i), \delta(\lambda_j)\}) + 1$, if λ was derived from λ_i and λ_j via `CondDet`. The term weight of a fact λ is two times the number of function symbols plus the number of variables occurring in λ. The heuristic LevFW—an extension of LevW—facilitates the prover to focus on certain function symbols. In order to calculate the term weight function symbols are not weighted with the value 2 any longer but each function symbol is associated with a special value given by the user.

The learning heuristics of [Fu96] use these heuristics as a basis and improve them by *re-enacting* a source proof or by learning important *feature values* for terms. We utilized the heuristic FeatW which employs the second alternative.

Since it is necessary for distributed theorem proving that the provers generate many different facts we developed some new heuristics following other ideas. One idea is to take the goal of the proof attempt into account by choosing either facts that are "similar" to the goal or facts whose *tail* is similar to the goal. The tail of a fact λ is defined in analogy to [Wo90], i.e. the tail of λ is the second argument of the left most occurrence of the function f in λ. Several ideas for computing a measure for similarity are presented in [DF94]. For the CLCH approach we found structural similarity quite useful. It is measured by comparing occurrences and nesting of function symbols of two terms. The use of structural similarity results in two good heuristics (called on_{fact} and on_{tail}). We recognized that the other heuristics of [DF94] were useful in TEAMWORK runs but not for CLCH due to their high specialization.

3.4.2 The Referees

In section 3.1 we have already mentioned that a lot of possibilities exist how selection processes can be organized. On the one hand it is possible to use only local knowledge for selecting of facts, on the other hand local knowledge can be extended with further knowledge. Such further knowledge can concern the heuristic or the current needs of the receivers, or even both. As we have discussed before, utilizing knowledge about the current needs of a receiver is only necessary if the quality of the used heuristics is rather low. Then, we need very sophisticated (and time-consuming) criteria in order to determine facts well-suited for the receiving provers. Since we are able to employ very powerful heuristics in the DiCoDe system which fall back on learning techniques we can be sure that they activate a lot of facts possibly well-suited for a receiving heuristic. Therefore we can resign the use of knowledge about the current needs of the receivers and select facts more efficiently. Thus, we do not employ receive-referees but only send-referees. In the send-referees we used local knowledge as well as knowledge about the heuristic of the receivers.

As already stated such a referee in the CLCH approach consists of a pair (S, φ). In the experiments of section 4 we used only one realization of S, namely $S(\lambda) = true$ iff λ was generated in the last working phase. So we limited the eligible facts to the same facts that referees for TEAMWORK can choose from.

In order to select facts φ falls back on certain judgement functions as previously described. There are two judgement functions for referees in DiCoDe. One is of type ψ_S and is simply called $\psi_S^{\mathcal{CD}}$. The other one is of type ψ_H and therefore called $\psi_H^{\mathcal{CD}}$. φ employed only one function for the selection, i.e. it either utilized $\psi_S^{\mathcal{CD}}$ or $\psi_H^{\mathcal{CD}}$ and selected such facts judged with the highest value w.r.t. $\psi_S^{\mathcal{CD}}$ or $\psi_H^{\mathcal{CD}}$, respectively.

Definition 3.1 The judgement function $\psi_S^{\mathcal{CD}}$
Let λ be a fact generated by a prover, $del(\lambda)$ the number of facts that could be subsumed by λ (using `Subsum`*) and $gen(\lambda)$ the number of applications of* `CondDet` *λ was involved*

in. Let further be f_{gen} and f_{del} real parameters. Then the value of λ according to the judgement function $\psi_S^{\mathcal{CD}}$ is

$$\psi_S^{\mathcal{CD}}(\lambda) = f_{gen} \cdot gen(\lambda) + f_{del} \cdot del(\lambda)$$

Since subsumption obviously reduces the branching factor of the search for a proof while applications of `CondDet` result in more potential facts f_{gen} should have a negative value and f_{del} a positive one (remember that facts with highest value are selected).

Definition 3.2 The judgement function $\psi_H^{\mathcal{CD}}$
Let A_P be the set of facts generated by a prover P (during a working phase) and R the receiver of the facts to be selected (with activation heuristic $\mathcal{H}_R$). Let further be $R^{max} = \max(\{\mathcal{H}_R(\lambda) | \lambda \in A_P\})$ and $R^{min} = \min(\{\mathcal{H}_R(\lambda) | \lambda \in A_P\})$. For a fact λ let R^{anc} be the maximal value $\mathcal{H}_R$ of all ancestors of λ from A_P, i.e. of the $\lambda' \in A_P$ that were needed to infer λ. Then the value of λ according to the judgement function $\psi_H^{\mathcal{CD}}$ is

$$\psi_H^{\mathcal{CD}}(\lambda) = \begin{cases} \frac{R^{max} - \mathcal{H}_R(\lambda)}{R^{max} - R^{min}} + \left(1 - \frac{R^{max} - R^{anc}(\lambda)}{R^{max} - R^{min}}\right) & ; R^{max} \neq R^{min} \\ 0 & ; otherwise \end{cases}$$

So, $\psi_H^{\mathcal{CD}}$ prefers those facts that have a small weight with respect to the selection heuristic of the receiving prover (first part of the sum) and that have additionally at least one ancestor with a high weight (second part of the sum). As already stated this last part is necessary in order to have a high probability that a selected fact was not already generated by the receiving prover.

There are as many instantiations of $\psi_H^{\mathcal{CD}}$ as selection heuristics exist in DiCoDe. Obviously, the results of an instantiation are only used for one receiving prover. We have chosen to normalize the values of $\psi_H^{\mathcal{CD}}$ (both parts of the sum compute always a value between 0 and 1) in order to compare values of several working phases in later evaluations.

3.4.3 Completeness of DiCoDe using CLCH

Although cooperation and distribution offer substantial improvements for automated theorem provers they can also cause some theoretical problems concerning completeness of the resulting systems. As pointed out in [AD93] a generating prover employing fair selection strategies may become incomplete if it receives new facts that are handled with priority thus contributing new facts to the set of potential facts. A stronger condition than fairness is needed that in [AD93] was called team-fairness but that more generally can be called *fairness despite disturbances*. Fortunately, most fair selection strategies are also fair despite disturbances.

In [AD93] it is shown that a system based on the Teamwork method is complete if a team-fair expert is infinitely often judged best expert of a working phase. Since the CLCH approach does not force the provers to a common start state after each cooperation phase completeness can be assured by using at least one prover with a heuristic that is fair despite disturbances.

3.5 The CLCH approach for Superposition

In the following section we want to show how CLCH can also be applied to already existing provers that were not especially developed for the use in a network of cooperating provers. Therefore, we applied CLCH to the superposition based theorem prover SPASS (see [WGR96]). This choice is motivated by the fact that—in opposite to the condensed detachment prover DiCoDe—SPASS employs a very general calculus enabling it to solve proof problems in full first order logic with equality. Furthermore, the sources of SPASS are easily available (see [WGR96]) and we could hence integrate code into this program. Thus, it is possible to implement internal referees instead of external ones which increases the efficiency and allows for the use of more knowledge.

Because of the fact that the implementation of the Teamwork cooperation scheme would in fact require a re-implementation we only integrated the CLCH scheme into SPASS.

3.5.1 The Control Heuristics

SPASS does not offer different control heuristics to a user but activates facts with the help of a fixed control heuristic W. W computes a weight of a clause as the sum of the weights of its literals. The weight of a literal L is two times the number of function and predicate symbols plus the number of variables occurring in L. Since we need different heuristics which have to be coupled in a network we added the heuristic FW to SPASS. FW—an extension of W—facilitates the prover to focus on certain function or predicate symbols. In analogy to heuristic LevFW—developed for DiCoDe—function or predicate symbols are not weighted with the value 2 any more but with a special value the user can assign to each symbol.

All in all we can say that in comparison to DiCoDe the implemented heuristics are quite primitive. Note, that no sophisticated heuristics, e.g. employing learning, are at SPASS' disposal.

3.5.2 The Referees

Because of the fact that we are not able to employ heuristics that are as good as in the area of condensed detachment it is even more important as in the previous section to design good referees that are able to select facts that are needed by the receivers. Therefore, the selection of facts employing a maximum of knowledge seems to be the right way to cope with this issue: A send-referee of each agent could select a rather high number of facts utilizing local knowledge as well as knowledge about the heuristics of the receivers. Thus, the probability increases that necessary facts are among them. Then, an individual receive-referee of each agent selects out of the facts—received from the send-referees of the other agents—facts that are considered to be important w.r.t. the current search state of its prover. In the sequel, we will describe how send- and receive-referees can be designed for superposition based theorem provers.

Send-Referees: As we have mentioned before a send-referee consists of a pair (S, φ). In our experiments we employed a predicate S realized as follows: $S(\lambda)$ holds if λ is not an axiom, was not selected from φ in an earlier cooperation phase, and was not received from the prover in an earlier phase. Thus, we avoid at least that facts are sent to another prover that are obviously in its system.

The selection function φ selects facts that are judged with a high value by certain judgement functions. We describe three judgement functions well-suited for the selection of facts. For each of the types ψ_S, ψ_G, and ψ_H we have developed one function. In order to select facts and to realize φ we employed each function for determining a certain percentage of good facts w.r.t. their judgement.

The function $\psi_S^{\mathcal{SP}}$ is of type ψ_S and counts—in analogy to section 3.4—inferences a fact was involved in to measure its success during the proof attempt so far. We restrict ourselves to the inference types superposition, general resolution, rewriting, and subsumption in order to reduce the number of parameters. $\psi_S^{\mathcal{SP}}$ is then defined in analogy to section 3.4:

Definition 3.3 The judgement function $\psi_S^{\mathcal{SP}}$
Let λ be a fact generated by a prover, $del(\lambda)$ the number of facts that could be subsumed by λ, $rew(\lambda)$ the number of facts that could be rewritten with λ, $res(\lambda)$ the number of general resolution steps λ was involved in, and $sup(\lambda)$ the number of applications of superposition steps λ was involved in. Let further be f_{del}, f_{rew}, f_{res}, and f_{sup} real parameters. Then the value of λ according to the judgement function $\psi_S^{\mathcal{SP}}$ is

$$\psi_S^{\mathcal{SP}}(\lambda) = f_{sup} \cdot sup(\lambda) + f_{gen} \cdot gen(\lambda) + f_{rew} \cdot rew(\lambda) + f_{del} \cdot del(\lambda)$$

We consider contraction inferences (subsumption and rewriting) as positive inferences and weight these with a positive factor, expansion inferences are considered to be negative.

The judgement function $\psi_G^{\mathcal{SP}}$ tries to estimate the success of a fact with the help of syntactic criteria. If a fact λ is quite general and has a "flat" structure it is possibly well-suited to subsume a lot of other ones. Furthermore, it is possibly not often involved in many applications of the superposition rule because there are not so many positions one can overlap in. $\psi_G^{\mathcal{SP}}$ prefers therefore facts that are quite small and have a flat structure: If a clause $C = \{L_1, \ldots, L_n\}$ is to be judged, $\psi_G^{\mathcal{SP}}(C) = -\sum_{i=1}^{n} \gamma(L_i, 0)$. γ is defined as:

$$\gamma(t,d) = \begin{cases} 1 + d & ; t \text{ is a variable} \\ 2 + d + \sum_{i=1}^{m} \gamma(t_i, d+1) & ; t \equiv f(t_1, \ldots, t_m) \end{cases}$$

One can see that occurrences at deeper positions are penalized more than occurrences at higher positions.

Function $\psi_H^{\mathcal{SP}}$ tries to select facts that are especially of need of the receiver. The judgement function $\psi_H^{\mathcal{SP}}$ is defined analogously to the function $\psi_H^{\mathcal{CP}}$ we have presented in section 3.4. The only difference is that the set A_P is given as the set of facts of a prover P that pass the filter S.

Receive-Referees: We consider two judgement functions for a receive-referee, functions $\psi_{SG}^{\mathcal{SP}}$ and $\psi_U^{\mathcal{SP}}$ of types ψ_{SG} and ψ_U, respectively. In analogy to the send-referees, each of this functions is responsible for the selection of a certain percentage of the facts received from the send-referees.

Function $\psi_{SG}^{\mathcal{SP}}$ tries to estimate if a clause can contribute to the solution of a certain (sub-)goal. Since we employ the superposition calculus we do not attempt to break a goal into different (sub-)goals. But each generated fact can be considered to be a new goal that has to be refuted. Therefore, $\psi_{SG}^{\mathcal{SP}}$ estimates if a fact λ possibly contributes to the refutation of an active fact λ' in the system of the receiver. Because of the fact that a receive-referee does not perform inferences by itself simple criteria have to be used in order to perform such an estimation. We assume that a fact λ contributes to the refutation of an active fact λ' if we can perform a resolution step with λ and λ'. However, a resolution step does not necessarily lead to a derivation of the empty clause but the clause length of the resulting clause can even increase. Hence, we assume that λ only contributes to a refutation of λ' if at least one resolvent of λ and λ' exists which length is shorter than the length of λ or λ'. In order to reduce the amount of computation needed to check this we use, more exactly, the following heuristic: If λ is a unit and resolves with λ' we assume it to be contributing to the refutation of λ'. If λ is not a unit clause we consider it to be only contributing to a proof of λ' if this clause is a unit and has a resolvent with λ. Furthermore, it is possible to take the length of the non-unit clauses into account: If we resolve a short clause it could be easier to derive the empty clause as if we resolve new clauses with very long ones.

The definition of $\psi_{SG}^{\mathcal{SP}}$ can be given as follows. Let the clause λ to be judged be a unit clause. Let λ' be an element of the active facts of the receiver, let $r(\lambda, \lambda')$ be the number of different resolvents from λ and λ'. Then we define

$$\nu(\lambda, \lambda') = \begin{cases} \infty & ; |\lambda'| = 1, r(\lambda, \lambda') > 0 \\ \frac{r(\lambda,\lambda')}{|\lambda'|} & ; \text{otherwise} \end{cases}$$

Otherwise, if λ is not a unit clause, we define

$$\nu(\lambda, \lambda') = \begin{cases} 0 & ; \lambda' \text{ is not a unit clause} \\ \frac{r(\lambda,\lambda')}{|\lambda|} & ; \text{otherwise} \end{cases}$$

Let Λ be the set of active facts of the receiver. By utilizing ν we define

$$\psi_{SG}^{\mathcal{SP}}(\lambda) = \sum_{\lambda' \in \Lambda} \nu(\lambda, \lambda')$$

The judgement function $\psi_{U}^{\mathcal{SP}}$ judges a fact λ w.r.t. its usefulness in the inference process starting with the current system of active facts of the receiver. A simple method in order to do this would be to perform some inferences and observe if λ takes often part in contracting but not in expanding inferences. This kind of judgement, however, is very time-consuming. Hence, we simplify this method and only count how often λ will be involved immediately in the subsumption rule if we integrate it, i.e. we count how many active facts can be subsumed by utilizing λ.

4 Experiments

In the sequel, we describe some experiments with the CLCH approach that we have performed with the condensed detachment prover DiCoDe and the superposition prover SPASS. Since DiCoDe is able to employ beyond CLCH also the Teamwork method our main interest here is to compare both cooperation schemes. We are especially interested in the question whether the simple scheme of CLCH provides enough cooperation among different provers.

Because of the fact that we are not able to let SPASS work in a Teamwork modus we investigated other topics in these experiments. In this area we investigate by some experiments whether CLCH is able to improve the standard setting of a prover that is not developed considering the idea of cooperation. Especially, we introduce a method to generate different cooperating heuristics automatically that are able to clearly outperform the standard setting of SPASS.

4.1 Experiments with Condensed Detachment

In this section we will analyze the performance of the CLCH approach in the area of condensed detachment. In particular, we will provide an experimental comparison of our CLCH approach with the Teamwork method, our best sequential heuristics, and the renowned theorem prover Otter (see [Mc94]). The test problems this comparison is based on stem from experiments by McCune and Wos (see [MW92]) with Otter. The problems can also be found in the TPTP library (see [SS94]), version 1.2.1, namely in the LCL domain. We use the names the problems have been assigned in the TPTP.

The prover CoDe and therefore also DiCoDe were developed to solve exclusively problems that can be tackled using the condensed detachment calculus. Since Otter has to use first-oder axiomatizations (but employs hyper-resolution, which results in less intermediate results) one might argue that DiCoDe has a small advantage (now that DiCoDe uses indexing techniques). But since we are mainly interested in comparing CLCH with Teamwork and the best heuristics the results of Otter are included to emphasis that we are not dealing with small and easy to solve problems here.

As already stated DiCoDe contains very strong heuristics based on concepts for learning from previous proof experiences. Therefore, there is always at least one heuristic capable of solving one of our test problems and, as table 1 demonstrates, these heuristics are quite efficient. This means that the potential for improvement by cooperation is not very high if the best heuristic for a problem is provided. But note that finding this best heuristic (or at least a good one) may involve several proof attempts which more than outweighs the use of several computers by DiCoDe either using CLCH or Teamwork.

Since there are several differences between CLCH and Teamwork choosing appropriate settings for our experiments was not easy. It was our goal to make the settings in CLCH mode and in Teamwork mode as comparable as possible. Therefore, the same heuristics (experts) are used and also the same referees. Unfortunately, this means that the flexibility provided by the set of (S,φ)-pairs in CLCH is not given anymore

problem	used heuristics		referees	CLCH	TEAMWORK	best expert	LevW	OTTER
LCL002-1	on_{fact} ,	on_{tail}	$\psi_H^{\mathcal{CD}}$	21.7	24.9	58.5	–	516
LCL003-1	on_{fact} ,	FeatW	$\psi_S^{\mathcal{CD}}$	55.5	91.9	91.4	–	449
LCL017-1	on_{fact} ,	FeatW	$\psi_H^{\mathcal{CD}}$	42.7	51.8	51.3	51.3	281
LCL040-1	LevW ,	LevFW	$\psi_H^{\mathcal{CD}}$	8.0	9.8	14.2	–	16
LCL054-1	on_{fact} ,	on_{tail}	$\psi_H^{\mathcal{CD}}$	19.3	29.1	34.7	–	Fail
LCL058-1	LevW ,	LevFW	$\psi_H^{\mathcal{CD}}$	17.6	33.6	52.9	52.9	423
LCL060-1	LevW ,	LevFW	$\psi_H^{\mathcal{CD}}$	6.5	29.7	36.9	56.1	447
LCL061-1	FeatW ,	FeatW	$\psi_H^{\mathcal{CD}}$	283.2	448.3	446.6	–	Fail
LCL071-1	FeatW ,	FeatW	$\psi_H^{\mathcal{CD}}$	4.8	6.5	29.1	–	511
LCL085-1	on_{fact} ,	on_{fact}	$\psi_H^{\mathcal{CD}}$	110.6	90.3	110.3	1200.5	2172
LCL097-1	on_{fact} ,	on_{tail}	$\psi_H^{\mathcal{CD}}$	8.8	9.3	40.5	44.0	2
LCL114-1	FeatW ,	FeatW	$\psi_H^{\mathcal{CD}}$	8.4	9.5	25.1	357.1	2035
LCL116-1	FeatW ,	FeatW	$\psi_H^{\mathcal{CD}}$	20.8	24.8	32.8	–	2041
LCL119-1	on_{fact} ,	FeatW	$\psi_H^{\mathcal{CD}}$	100.7	67.8	128.9	–	362

Table 1: CLCH vs. TEAMWORK vs best heuristic in LCL domain

(and the number of computers has to be limited to two in order to allow referees of type $\psi_H^{\mathcal{CD}}$ that can also be employed in TEAMWORK, using the heuristic of the winner). On the other hand using the same heuristics means that the adaptation capabilities of TEAMWORK cannot come into play since each expert is always active.

This way, the main criteria influencing the results of our experiments are the overhead caused by the two cooperation concepts and the utility of employing a survival-of-the-fittest strategy (as realized in TEAMWORK) or not (as in CLCH). We used for the experiments two SPARCstation ELC running Sun OS 4.1. The runtimes for OTTER stem from [MW92] and were obtained on a SPARCstation 1+, a machine comparable to ours. "Fail" denotes that OTTER was not able to solve the problem within four hours. Note that OTTER was not used in its auto-mode but we list here the results of the best of up to six different heuristics. The entry "–" denotes that the respective heuristic was not able to solve the problem within 2000 seconds. All runtimes in table 1 are given in seconds.

Table 1 shows that both cooperation concepts can improve the performance of DiCoDe. For 12 of the 14 problems the CLCH approach resulted in better runtimes than the TEAMWORK method, for some of the problems (for problem `LCL061-1`, `LCL003-1` or `LCL058-1`) the improvements are quite substantial. But, as expected, there are problems (`LCL085-1`, `LCL119-1`) for which TEAMWORK results in a better performance. An analysis of these problems revealed that the better performance of TEAMWORK is due to the common start state after each team meeting. In both problems one expert uses the state of the other winning expert to find a proof more quickly. But DiCoDe was also able to solve these problems in CLCH mode although the runtime is nearly equivalent to the runtime of the best heuristic for these problems. Note that there are also some problems for which the TEAMWORK modus only produced a run nearly equivalent to the best expert.

The analysis of the problems for which the runtimes of CLCH and TEAMWORK are nearly equal showed that DiCoDe generated for both modi the same proofs and nearly the same runs. Due to the lesser overhead, however, CLCH was faster. In case of the substantial improvements by CLCH, Teamwork chooses the wrong winner which resulted in the already described additional computation until this mistake was corrected. But note that Teamwork nevertheless often outperforms the best heuristic working alone.

If we compare the results of DiCoDe in CLCH mode with the best sequential heuristics (note that "best heuristic" means that they are the best we found; but the comparison with Otter shows that these heuristics that very often use learned knowledge from previous proof attempts are quite good), then we can observe that for 12 problems there are sometimes quite substantial improvements due to the cooperation of the heuristics. We have an improvement of a factor of 6.1 and a factor 5.7 when solving problems `LCL071-1` and `LCL060-1`, respectively. Furthermore, the improvement factors are greater than 2 for six problems.

Note that using CLCH different learning heuristics (using either different source proofs or different aspects of the same source proof) can cooperate allowing to find proofs faster (problems `LCL061-1`, `LCL071-1`, `LCL114-1`, `LCL116-1`). Thus our goal, to improve already good but not quite good enough heuristics by cooperation with other good ones, is fulfilled.

Finally, the experiments show that if very good heuristics are available it is sufficient to employ only one send-referee in order to select facts: There are enough facts among the selected ones which really allows the receiver to find the proof considerably faster. Moreover, the overhead caused by the cooperation phases is very small.

4.2 Experiments with Superposition

In the previous section we particularly considered the differences between CLCH and Teamwork by means of a prover which was developed for realizing such cooperation schemes. In this section we want to show that CLCH is able to couple incarnations of a prover—the superposition based prover SPASS—which was not developed considering the idea of cooperation.

We had to choose again challenging problems for accomplishing this task. To this end, we tackled with SPASS problems stemming from the CADE-13 ATP system competition ([SS96]). Among these problems we selected those from the categories "unit equality" and "mixed" that SPASS was only able to solve running at least 8 seconds on a SPARCstation-20 (medium and hard problems).

As described in section 3.5, SPASS offers only a few different heuristics. Moreover, there is not much experience about the appropriateness of heuristics for certain problems. Since it is not our task to search for heuristics that are suitable for cooperation but we want to show that by means of CLCH speed-ups can easily be achieved, we decided to generate heuristics automatically. In analogy to before we let two heuristics cooperate, each running on a SPARCstation-20. One of the heuristics was the SPASS "standard heuristic" W (see section 3.5), the other was a heuristic of type FW generated in the following manner: Function symbols occurring in the axiomatization but not in

problem	SPASS	$\mathcal{H}_1$	$\mathcal{H}_2$	CLCH	S_{st}	S_{best}	S_{av}
LCL196-1	292.4	292.4	311.7	83.8	3.5	3.5	3.6
LCL163-1	10.0	10.0	11.9	7.5	1.3	1.3	1.4
GRP048-2	23.3	23.3	17.4	8.7	2.5	2.0	2.2
GRP148-1	951.2	307.9	253.1	184.7	5.1	1.4	1.5
GRP169-1	80.1	15.7	29.2	13.7	5.8	1.2	1.6
GRP169-2	56.1	56.1	26.2	9.7	5.8	2.7	4.2
GRP174-1	8.0	8.0	8.3	5.8	1.4	1.4	1.4
RNG018-6	639.9	639.9	199.7	152.3	4.2	1.3	2.8
NUM009-1	8.1	8.1	6.3	2.6	3.1	2.4	2.8

Table 2: CLCH for superposition theorem proving

the goal clauses were weighted with the value 5, the other symbols with the value 2. Only at examples `GRP169-1` and `GRP148-1` we deviated from this method. Because of the fact that the standard heuristic has only a very weak performance we employed another heuristic of type FW.

In each of our experiments we chose the same fixed parameterization of the judgement functions needed by the receive- and send-referees. As we have mentioned in the previous chapter we let each judgement function of a referee select a certain percentage of facts. Each of the three judgement functions of a send-referee selected 33% of the facts to be sent to the other agents. The receive-referee eventually integrated 60% of these facts into the system of its associated theorem prover. Each of the judgement functions $\psi^{\mathcal{SP}}_{SG}$ and $\psi^{\mathcal{SP}}_{U}$ selected 30% of the facts received from the send-referees.

It is to be emphasized that we employed a very simple implementation in order to exchange facts: We used the standard file mechanism of UNIX for the exchange of facts, i.e. the send-referees wrote good facts into a file that was later read by the receive-referees. Thus, we chose a rather inefficient but very simple scheme to let the referees communicate. We used files instead of UNIX sockets in order to show the minimal requirements (in terms of changes to the program: each prover has procedures to write facts into files) of the CLCH approach. Even with such a simple and inefficient implementation rather high speed-ups are possible as described shortly.

Table 2 shows the results we obtained when tackling the proof problems. We compare the runtimes of the standard setting of SPASS (column marked with SPASS), the runtimes each of the coupled heuristics needs when solving the problem alone (columns 3 and 4), and the time needed when coupling the heuristics via CLCH. The remaining three columns present speed-ups in comparison to the standard setting of SPASS, the best of the coupled heuristics, and the average of the coupled heuristics. It shows that we can achieve rather high speed-ups when we utilize CLCH instead of working with SPASS in its default setting. In almost all cases the speed-up is greater than 2, we achieve even super linear values of 5.8. This is very satisfactory, especially if we remember that the overhead caused by the simple implementation of CLCH is quite high. S_{best} shows the gain of efficiency compared with the best of the coupled heuristics.

This shows the potential of CLCH to improve even good heuristics when coupling them with worse ones. Naturally the speed-ups are not as high as before. Nevertheless, they are satisfactory. S_{av} is the speed-up in comparison to the arithmetical middle of the runtimes needed by $\mathcal{H}_1$ and $\mathcal{H}_2$. This gives a hint about the gain of efficiency in comparison to an arbitrary choice of the automatic generated heuristics. As one can see in the table these speed-ups range from the value 1.4 to the value 4.2. Hence, CLCH clearly outperforms an arbitrary choice of automatic generated heuristics.

All in all we can say that the results are quite satisfactory. Nevertheless, they are not as good as in the area of condensed detachment. There are three reasons for this: On the one hand the learning heuristics employed in DiCoDe need often only a few results to conclude the proof that can be submitted from the other provers. Since this is not the case for SPASS the results are worse. Moreover, to deal with this problem we had to develop more sophisticated referees and had to use both send- and receive-referees. But this, on the other hand, causes more overhead by the cooperation. Finally, communication through files is an inefficient method and causes a lot of overhead. Utilizing a faster communication technique, e.g. UNIX sockets, instead of files is therefore an easy way to improve the performance.

5 Conclusion and Future Work

We have presented an approach for coupling several incarnations of a generating theorem prover that use different search-guiding heuristics. The cooperation of these incarnations is achieved by periodically interchanging generated results that are selected by so-called referees. We presented a lot of different kinds of referees utilizing different knowledge for the selection of facts. Furthermore, we introduced an architecture that is able to integrate all these kinds of referees efficiently into the selection process.

This concept is far easier to implement than the Teamwork method and also offers some new possibilities with respect to judging results. Also the lack of a central control results in a more fault tolerant system. This is paid for by loosing the survival-of-the-fittest principle of Teamwork and the adaptation to the given problem by reactive planning.

However, our experiments with a condensed detachment prover showed that our approach achieves cooperation between control heuristics resulting in much shorter runtimes. Especially, if some of the used heuristics utilize learned (and therefore often not sufficiently complete) knowledge the profits of the cooperation are very high. Moreover, the results in coupling different incarnations of the superposition based prover SPASS showed that CLCH is even well-suited for simple heuristics that are based on syntactical criteria.

Future research will be concerned with two directions. One direction is to develop a cooperation concept combining the advantages of CLCH and Teamwork. Although we cannot expect that such a concept will be as easy to implement as CLCH it can offer some interesting possibilities. These are the survival of several good search states after a team meeting and the use of referees that are more concerned with the receiving experts. Furthermore it should be possible to employ the reactive planning capabilities of TEAMWORK.

But the CLCH approach offers also the possibility to use different theorem provers using different calculi as long as they produce facts that can be interchanged. While TEAMWORK has to prefer one kind of prover with one calculus CLCH can handle all these provers as equals. The more flexible referee concept allows for meeting the needs of quite different provers within one proof attempt.

References

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

[BG94] **Bachmair, L.; Ganzinger, H.**: *Rewrite-based equational theorem proving with selection and simplification*, Journal of Logic and Computation, 4(3), 1994, pp. 217–247.

[CMM90] **Conry, S.E.; MacIntosh, D.J.; Meyer, R.A.**: *DARES: A Distributed Automated Reasoning System*, In Proc. AAAI-90, 1990, pp. 78–85.

[De90] **Dershowitz, N.**: *A maximal-Literal Unit Strategy for Horn Clauses*, Proc. 2nd CTRS, Montreal, LNCS 516, 1990, pp. 14–25.

[De95] **Denzinger, J.**: *Knowledge-Based Distributed Search Using Teamwork*, Proc. ICMAS-95, San Francisco, AAAI-Press, 1995, pp. 81–88.

[DF94] **Denzinger, J.; Fuchs, M.**: *Goal-oriented equational theorem proving using teamwork*, Proc. 18^{th} KI-94, Saarbrücken, LNAI 861, 1994, pp. 343–354.

[DF96] **Denzinger, J.; Fuchs, D.**: *Referees for Teamwork*, Proc. FLAIRS '96, Key West, FL, USA, 1996, pp. 454–458.

[DK96] **Denzinger, J.; Kronenburg, M.**: *Planning for Distributed Theorem Proving: The Teamwork Approach*, Proc. KI-96 (German annual conference on AI), Dresden, GER, LNAI 1137, 1996, pp. 43–56.

[DS96a] **Denzinger, J.; Schulz, S.**: *Recording and Analyzing Knowledge-Based Distributed Deduction Processes*, JSC 21, 1996, pp. 523–541.

[DS96b] **Denzinger, J.; Schulz, S.**: *Learning Domain Knowledge to Improve Theorem Proving*, Proc. CADE-13, New Brunswick, NJ, USA, LNAI 1104, 1996, pp. 62–76.

[Fu96] **Fuchs, M.**: *Experiments in the Heuristic Use of Past Proof Experience*, Proc. CADE-13, New Brunswick, NJ, USA, LNAI 1104, 1996, pp. 523–537.

[FF97] **Fuchs, D.; Fuchs, M.**: CoDe*: A Powerful Prover for Problems of Condensed Detachment*, Proc. CADE-14, Townsville, Australia, 1997, to appear.

[Gr96] **Graf, P.**: *Term Indexing*, LNAI 1053, 1996.

[Lu70] **Łukasiewicz, J.**: *Selected Works*, L. Borkowski (ed.), North-Holland, 1970.

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

[MW92] **McCune, W.; Wos, L.**: *Experiments in Automated Deduction with Condensed Detachment*, Proc. CADE-11, Saratoga Springs, NY, USA, 1992, LNAI 607, pp. 209–223.

[Pe76] **Peterson, G.J.**: *An automatic theorem prover for substitution and detachment systems*, Notre Dame Journal of Formal Logic, Vol. 19, Number 1, January 1976, pp. 119–122.

[Sl93] **Slaney, J.**: *SCOTT: A Model-Guided Theorem Prover*, Proc. IJCAI '93, Chambery, FRA, 1993, pp. 109–114.

[SS96] **Sutcliffe, G.; Suttner, C.**: *ATP System Competition* held on August 1 in conjunction with CADE-13, New Brunswick, NJ, USA, 1996; Competition results available via WWW at the URL `http://wwwjessen.informatik.tu-muenchen.de/~tptp/CASC-13`.

[SS94] **Sutcliffe, G.; Suttner, C.; Yemenis, T.**: *The TPTP Problem Library*, Proc. CADE-12, Nancy, FRA, 1994, LNAI 814, pp. 252–266.

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

[We93] **Weidenbach, C.**: *Extending the resolution method with sorts*, Proc. IJCAI '93, Chambery, FRA, 1993, pp. 60–65.

[WGR96] **Weidenbach, C.; Gaede, B.; Rock, G.**: *SPASS & FLOTTER Version 0.42*, Proc. CADE-13, New Brunswick, NJ, USA, LNAI 1104, 1996, pp. 141–145.

[Wo90] **Wos, L.**: *Meeting the Challenge of Fifty Years of Logic*, Journal of Automated Reasoning 6, 1990, pp. 213–232.

[Wo95] **Wos, L.**: *Searching for Circles of Pure Proofs*, JAR 15, 1995, pp. 279–315.