DISCOUNT

A distributed and learning equational prover

JÖRG DENZINGER and MARTIN KRONENBURG
FB Informatik, Universität Kaiserslautern
67653 Kaiserslautern, Germany
{denzinge,kronburg}@informatik.uni-kl.de

STEPHAN SCHULZ
Institut für Informatik, Technische Universität München
80290 München, Germany
schulz@informatik.tu-muenchen.de

Abstract. The DISCOUNT system is a distributed equational theorem prover based on the teamwork method for knowledge-based distribution. It uses an extended version of unfailing Knuth-Bendix completion that is able to deal with arbitrarily quantified goals. DISCOUNT features many different control strategies that cooperate using the teamwork approach. Competition between multiple strategies, combined with reactive planning, results in an adaptation of the whole system to given problems, and thus in a very high degree of independence from user interaction.

Teamwork also provides a suitable framework for the use of control strategies based on learning from previous proof experiences. One of these strategies forms the core of the expert `global_learn`, which is capable of learning from successful proofs of several problems. This expert, running sequentially, was one of the entrants in the competition (DISCOUNT/GL), while a distributed DISCOUNT system running on two workstations was another entrant.

Key words: Automated theorem proving, competition, DISCOUNT, distributed theorem proving, reactive planning, learning

1. Introduction

The DISCOUNT system (DIStributed COmpletion UsiNg Teamwork) is a distributed proof system for equational reasoning. It was originally intended as a first experimental system for studying the teamwork method, a knowledge-based distribution concept for certain theorem proving procedures (and certain other search processes, see [4]). However, teamwork can (and in our implementation does) combine many different interesting concepts: cooperation and competition among control heuristics, reactive planning of proof attempts (resulting in an adaptation to a given problem), adequate representation and application of different kinds of knowledge, the use of automatically acquired control knowledge, and the use of goal-oriented heuristics in completion. The integration of these concepts has resulted in multiple versions of the system, some of which are quite capable of competing with other renowned proof systems in the field of equational deduction.

A system that uses the teamwork method has four types of components: experts, specialists, referees and a supervisor. *Experts* and *specialists* are the components actively working on the solution of a given problem. In the case of DISCOUNT most of them are (equational) theorem provers. They work independently for given periods of time, using their own method or their own view on the problem. All experts employ the same basic proving technique (unfailing completion). They only differ in the way they select facts for processing. Specialists, on the other hand, may employ any correct means to generate new equations, and can also support the supervisor in administrative tasks. A team has as many active experts and specialists as there are computers available. Each runs on its own computing node.

After the experts and specialists have worked for a set period, a *team meeting* takes place. In the first phase of a team meeting the work of all active experts and specialists is judged by *referees*. Each expert and specialist is judged by an individual referee. A referee has two tasks: measuring the overall progress of an expert or specialist in the last period (resulting in a *measure of success*), and selecting outstanding new results (equations). It runs on the same processor as the corresponding expert or specialist, and shares the same memory.

The results of the referees (measures of success and outstanding results) are collected by the *supervisor*. The supervisor determines the most successful expert and uses its complete search state as a base for a new working period. It also incorporates the outstanding results of the other experts and specialists into this state. The supervisor then determines the composition of the team for the next working period and broadcasts the new search state to all experts and specialists. To select the members of the new team the supervisor uses *reactive planning*. This process is based on both general knowledge about experts, specialists, referees, and domains of interest (the *long-term memory*), and on information gathered during the current proof attempt (the *short-term memory*), as for example the measures of success.

2. Architecture

DISCOUNT is based on the teamwork approach and contains many different components. This report concentrates on the components most relevant for the competition. Therefore we will only describe the inference rules and the basic completion algorithm, a particular heuristic employing learned knowledge (see [9] for a more in-depth discussion), and the reactive planning mechanism used by the supervisor (see [7]). Information about other experts can be found in [6], and [5] concentrates on referees.

2.1. The Calculus

DISCOUNT is a prover for pure unit-equality problems. The input for the prover consists of a set E of (implicitly universally-quantified) equations and a goal $u = v$ that may either be ground (universally quantified variables are eliminated by Skolemization) or contain variables (resulting from existential quantifiers).

The case of a ground goal is covered by conventional unfailing completion, as described by the inference system $\mathcal{U}$ from [2]. Provers based on unfailing completion work on the input equations E and the goal, and use a (ground) reduction ordering $\succ$ to restrict the application of equations and to guarantee termination of the rewriting process. Our inference system $\mathcal{G}$ is based on $\mathcal{U}$ and extends it to deal with goals containing variables.

In the following definition we use standard notations. $s \Rightarrow_R t$ denotes that s can be rewritten to t using a rule from R, $s \Rightarrow_{R(E)} t$ denotes a rewrite step with either a rule from R or an orientable instance of an equation from E, and $s \Leftrightarrow_{E\cup R} t$ denotes that s can be transformed into t by applying one rule or equation in either direction. Note that equations are symmetric, that is $s = t$ also represents $t = s$.

Definition 1. (The inference system $\mathcal{G}$) Let $\succ$ be a reduction ordering. The inference system $\mathcal{G}$ works on triples (E, R, G), where E is a set of equations, R a set of rules (orientable equations), and G a set of goals. It consists of the inference system $\mathcal{U}$ and the rules

G1: *Subsume goal*: $(E, R, G \cup \{s = t, u = v\}) \vdash_{\mathcal{G}} (E, R, G \cup \{s = t\})$, if there is a σ such that $u \equiv \sigma(s)$ and $v \equiv \sigma(t)$.

G2: *Cover goal*: $(E, R, G \cup \{s = t\}) \vdash_{\mathcal{G}} (E, R, G \cup \{s = t, u = v\})$, if there is a σ such that $\sigma(s) \Rightarrow_{R(E)} u$ and $v \equiv \sigma(t)$ ($u = v$ is a *critical goal*).

G3: *Success*: $(E, R, G \cup \{s = t\}) \vdash_{\mathcal{G}}$ Success, if s and t are unifiable.

For more details, please refer to [1] and [3].

The system $\mathcal{G}$ induces a refutation procedure. An arbitrarily quantified goal $Q_1x_1....Q_mx_m.u' = v'$ is negated and Skolemized, resulting in a formula $u \neq v$ (where remaining variables are now universally quantified). If we can show that there is an instance σ, such that $\sigma(u) =_E \sigma(v)$, then the negated goal is inconsistent with the axioms, and hence the original formula $Q_1x_1....Q_mx_m.u' = v'$ is a logical consequence of E. Provided that the selection of inference steps is fair, one can show that the rule *Success* will be applied if and only if $Q_1x_1....Q_mx_m.u' = v'$ is a consequence of E (see [3]).

As stated above, all of DISCOUNT's experts use the same basic algorithm. In order to allow a good control of the search process, all possible critical pairs and critical goals with respect to the current system (E, R, G)

are stored in two lists, CP and CG (which are ordered with respect to the values a selection function assigns to the elements; see 2.2). This results in the following algorithm. It starts with empty sets E,R, and G, the initial goal in CG, the initial equations in CP, and a reduction ordering $\succ$.

repeat
  select an element of CG or CP and remove it from the list
  normalize this element
  if the element is neither trivial nor subsumed **then**
    interreduce the sets E,R and G
    add the element to the appropriate list (E,R or G)
    compute all non-trivial critical pairs and goals with the element
until either CP and CG are empty ("NO") or *Success* ("YES").

Normalization and interreduction use the simplification rules as long as any is applicable. This process always terminates. Critical pairs and critical goals are processed in a fixed ratio. When deciding which equation to select for processing (the most crucial choice for the performance of the prover), there are different preferences for critical goals and critical pairs, resulting in different evaluation functions for the sets CG and CP in an expert.

2.2. The Control

As already stated, there are two different levels of control in DISCOUNT. On the tactical level, an expert controls the selection of critical pairs and critical goals. On the strategical level, the supervisor controls which experts (and specialists) are allowed into the teams working on the problem.

DISCOUNT has many different generic experts, i.e. experts described by parameterized evaluation functions. In addition to functions that count function symbols and variables, we have functions that allow linear and polynomial interpretations of function symbols. We also use functions that only interpret the greater term with respect to the reduction ordering $\succ$, or only select equations that can be oriented.

Teamwork allows us to employ specialized evaluation functions that may be unsuitable as general purpose heuristics, but can significantly speed up the proof process in certain phases. Among those are functions that determine the similarity of terms, either for the two terms of a goal ($UnifGoal$) or for a critical pair and a goal [6]. Finally, we have developed evaluation functions that use knowledge learned from prior proofs.

2.2.1. *Acquisition of Search Control Knowledge*

The development of a good heuristic for a given domain, and even the selection of good parameters for the generic experts, is a difficult task. Moreover, even the best static heuristics still generate an enormous overhead when compared to the theoretical optimum. This indicates that it is difficult for humans to recognize all the important concepts for a good evaluation function from the large amount of data even a few proof attempts generate.

These reasons motivated us to research into the possibility of automatically learning search control knowledge from successful proof attempts. One of our approaches resulted in the evaluation function `global_learn`. It is based on the memorization of *patterns* of positive examples (equations contributing to one or more proofs). See [9] and [11] for information on other approaches.

The first step for the analysis of proof attempts is the recording of the reasoning process. We developed PCL (*Proof Communication Language*), a language that expresses the inference process as a list of individual steps. We also managed to get protocols of distributed proof runs without disturbing the delicate timing of the various components, and to restructure them into a single sequential listing of inferences (see [8, 10]). The complete protocol is striped of all steps not contributing to the proof. The result is used in our learning approach (which requires positive examples only).

To cut down on the amount of information and to minimize the overhead at runtime, the data gained from the proofs is preprocessed and then stored into a knowledge base for efficient access. Equations are generalized into *representative patterns* which are used as indices in the knowledge base.

Definition 2. (Representative Term Patterns) Let t be a term over some signature. Let $T = \{f_1, f_2, \ldots\}$ and $X = \{x_1, x_2, \ldots\}$ be two disjoint totally ordered sets of symbols. The *representative pattern* of t with regard to T and X is constructed by substituting the function symbols and the variables in t with symbols from T and X, respectively, in their order of appearance. Thus, the representative pattern of the term $t \equiv f(g(x), g(y))$ with respect to T and X would be $t' \equiv f_1(f_2(x_1), f_2(x_2))$. We call the representative pattern of t $pat(t)$.

The two orderings on T and X can be combined into a total ordering $>$ on $T \cup X$ by assuming $f_i > x_j$ for all $f_i \in T$ and all $x_j \in X$. This total ordering obviously induces a total lexicographical ordering $>_{pat}$ on term patterns as well. $>_{pat}$ is extended to a quasi-ordering on all terms by assuming $s \geq_{pat} t$ iff $pat(s) \geq_{pat} pat(t)$ for arbitrary terms s and t. Using this ordering, we can define representative patterns for term pairs (rules or equations).

Definition 3. (Representative Patterns for Equations) Assume T and X as above. The *representative pattern* for an equation $s = t$ is generated by orienting the equation according to $>_{pat}$ and then substituting the function symbols and the variables in $s = t$ with symbols from T and X as above. We again write $pat(s = t)$ to denote the representative pattern of $s = t$.

Patterns serve as an easily computable abstraction of equations and allow us to implement efficient knowledge retrieval mechanisms. The knowledge base consists of multiple sets of these patterns, annotated with information about the occurrence of the corresponding equations in previous proofs. For each equation we store the name of the source proof, the number of times it was used in this proof, and the distance (in inferences) of the first occurrence of this equation to the goal with its representative pattern.

2.2.2. *The Expert* `global_learn`

Currently, learned data is only used for guiding the critical pair evaluation. The `global_learn` expert uses pattern retrieval on the whole knowledge base to evaluate new critical pairs. It employs DISCOUNT's default, *UnifGoal*, for the selection of critical goals.

If `global_learn` is confronted with a new critical pair $s = t$, it tries to retrieve the annotations stored with $pat(s = t)$ in the entire knowledge base. If the pattern does not occur in the knowledge base, the equation gets a standard evaluation based on the symbol count, with a constant penalty added to the result. Otherwise, the evaluation is based on the retrieved data. The assigned value is a linear combination of the number of proofs the pattern appeared in, the total number of applications of the pattern in previous proofs, the average number of applications per proof, and a term comparing the average goal distance of the pattern in previous proofs and the distance from the goal in the current proof process. As many patterns occur in only a few proofs, the knowledge often leads to the same evaluation for different equations. To avoid these ties, we add the normal term weight.

In our experience, the resulting strategy is excellent at reproducing known proofs. Even if proofs from very different sources (different conventional strategies, goal-oriented strategies, distributed proofs, other learning strategies) are combined into one knowledge base, `global_learn` is able to reproduce nearly all of the inserted proofs. But it also has extrapolative power and can find proofs no other strategy (or combination of strategies in a team) can find. Generally, this extrapolative power becomes greater with an increased amount of knowledge.

2.2.3. *Reactive Planning by the Supervisor*

In this subsection we sketch the concepts the supervisor uses to control and plan the work of the whole team. For a more detailed description see [7]. We do not discuss the other tasks of the supervisor.

There are two parameters which can be tuned by the supervisor during a team meeting to control the overall search process: the *members of the next team* and the *length of the next working period*.

The major problem with planning techniques in the area of automated theorem proving is the general vagueness of the available information. Another problem is caused by the special role of the supervisor in the teamwork method. During the time it is working, all other components are inactive. Thus, special care has to be taken to overcome this potential bottleneck.

Our solutions to these two problems are based on three concepts. The first concept is *teamwork* itself, which allows planning on an abstract level adequate for dealing with vague information. This abstract level is represented by the experts and referees. The available knowledge is insufficient for planning at a lower level, e.g. that of inference steps. Due to the vagueness of the available information, one cannot assume that a generated plan will be successful. Consequently, it must be possible to revise faulty plans. The judgments of the referees support the detection of such plans. The supervisor is then responsible for the correction. This leads to a second concept: *reactive planning*. During each team meeting the supervisor tries to improve the current plan by combining long-term memory and short-term memory.

A third concept solves the bottleneck problem. We implemented the planning of the supervisor as an *anytime algorithm*. This means that it decides dynamically when to stop its planning. The algorithm guarantees that a plan is available at any time.

The realization of these three concepts is based on the ideas of *domains* and *plan maintenance*. A domain is a collection of information about a set of proof problems and is stored in the long-term memory. The description of a domain consists of the facts defining it, important lemmas, suitable strategies, and relations to other domains. To detect a domain we have implemented a *domain detection specialist*, which is always a member of the first team. This specialist reports all domains detected in the current problem. During the first team meeting the supervisor determines one domain it will concentrate on by using information about the relations between the reported domains. The strategies associated with this domain include three teams suitable for the domain in three different proof phases. This enables the supervisor to set up a *plan skeleton*. Furthermore, the information about detected domains is used to fill this plan skeleton with suitable experts.

During future team meetings, the supervisor tries to improve and to adapt this plan skeleton by performing the following steps: It first determines the conditions for stopping the planning process. Next it checks if the proof phase has changed and if the plan skeleton is still suitable. Finally, it determines the new team and the time of the next team meeting.

To select the experts for the next working period, the supervisor first categorizes all available experts into several groups. Then the experts of each group are judged according to a list of criteria, which combine information from both, long-term and short-term memory. Every time the experts of one group have been judged with respect to a new criterion, the supervisor checks if it is necessary to consider further experts and criteria. Whenever this planning process stops, the supervisor has a judgment for each expert, calculated as a weighted sum of the criteria considered so far. The experts with the best judgments are selected for the next team.

The length of the next working period is determined by modifying default values (stored in the domain description) according to the current performance of the team and the number of facts in the current search state.

2.3. The Competition Versions

Two different versions of DISCOUNT entered the CADE-13 ATP competition. A sequential version using the `global_learn` expert entered in the general hardware category as DISCOUNT/GL. The latest distributed system, DISCOUNT 3.0, running on two workstations, entered in the special hardware category, but was restricted to using static experts only.

The DISCOUNT/GL system was not optimized towards the subset of eligible TPTP problems, but trained on the set of all unit-equality problems from this problem library. The knowledge base contained data about 664 proofs for 226 of the 375 unit-equality problems. It was constructed using data from the static experts *AddWeight*, *Occnest*, and *MaxWeight*, Fuchs' analogy-based learning experts [11], and a small number of proofs found with the distributed system. About 20 additional proofs were found during a bootstrapping phase, i.e. by application of the pre-trained `global_learn` expert to unsolved problems in order to generate more training data.

As in the case of the DISCOUNT/GL system, DISCOUNT 3.0 was not tuned to the subset of eligible TPTP problems. The long-term memory consisted of 48 different experts, three simple statistical referees, and 29 domains. The knowledge represented in this long-term memory was obtained by manually exploiting proof attempts performed by several different teams and sequential experts.

3. Implementation

The DISCOUNT-system is implemented in ANSI-C, using the GNU C compiler, and was developed on SUN computers under SunOS 4.1.3. A port of DISCOUNT to Solaris is nearly complete. The basis of the distribution part is a secure real broadcast. The auxiliary tools for proof protocol processing and knowledge base maintenance are also written in ANSI C and can be compiled and executed on any UNIX system with standard utilities.

4. Performance in the Competition

In general, we think that the performance of both versions was quite satisfactory. One of the problems we encountered was the selection of a proper reduction ordering. DISCOUNT usually expects the user to provide a suitable ordering. For the competition we were forced to rely on a fast and simple automatic solution. We used a *lexicographic path ordering* and generated a total precedence on the operators using arity information and order of parsing. This resulted in suboptimal orderings for many domains. Moreover, as the input clauses were reordered for the competition, the resulting precedences did not always match the ones expected by some of DISCOUNT's knowledge-based techniques.

Both versions are relatively slow at startup. DISCOUNT/GL has to read its knowledge base and the distributed version has to start processes on the other computers and to build the communication structure.

The DISCOUNT/GL version was handicapped by the fact that time constraints did not permit us to reproduce and insert all of DISCOUNT's distributed proofs in the knowledge base. The new Waldmeister prover [12] also produces PCL, and thus could serve as a source for training examples. However, due to some problems in the implementation it was impossible to obtain Waldmeister proofs in time.

The distributed version of DISCOUNT faced a big problem in the competition, namely that the only available machines capable of broadcasting were SPARCstation Classic computers. Since the time allotted for a run was the same as for the other competitors (on SPARCstation 10 machines that are approximatively twice as fast), the number of team meetings during a run was not very high. We had to compensate for the slower machines by doubling the length of the working periods to produce enough results per period. Since the adaptation to a given problem needs several team meetings, the reactive planning process was significantly handicapped.

5. Conclusion

5.1. Strengths and Weaknesses of the System

DISCOUNT's main strength lies in the powerful heuristics and the synergetic effects achieved by the use of teamwork. Moreover, the implementation of PCL and the auxiliary tools dealing with proof protocols allow us to analyze proofs and to present them to human users in an easily understandable way.

The major weakness of the current DISCOUNT system is its basic inference engine. When the system was designed in 1991, most proof systems were still implemented in LISP, and neither indexing techniques nor structure sharing were widely used. Consequently, and because of its prototypical nature, DISCOUNT was build using straightforward solutions for term rewriting and storage. Especially for larger examples, systems like Waldmeister achieve a throughput that is orders of magnitude greater than DISCOUNT's.

5.2. Future Plans

The future developments we are interested in will concentrate on incorporating even more knowledge-based features into DISCOUNT. In the current version there are already several experts using learned knowledge (some focusing on one source problem with a source proof, in contrast to `global_learn`). These experts are already integrated in the database for the reactive planning process. The automatic integration of new experts and new domains into these databases is a second problem we are working on. Finally, we hope to use a much faster basic inference engine in the near future.

References

1. J. Avenhaus and J. Denzinger. Distributing Equational Theorem Proving. In *Proc. of the 5^{th} RTA, Montreal*, number 690 in LNCS, pages 62–76. Springer, 1993.
2. L. Bachmair, N. Derschowitz, and D.A. Plaisted. Completion Without Failure. In *Coll. on the Resolution of Equations in Algebraic Structures, Austin, 1987*. Academic Press, 1989.
3. J. Denzinger. *Teamwork: Eine Methode zum Entwurf verteilter, wissensbasierter Theorembeweiser*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, 1993. (German Language).
4. J. Denzinger. Knowledge-Based Distributed Search using Teamwork. In *Proc. of the ICMAS-95, San Francisco*, pages 81–88. AAAI Press, 1995.
5. J. Denzinger and D. Fuchs. Referees for Teamwork. In *Proc. 9^{th} FLAIRS, Key West*, pages 454–458, 1996.
6. J. Denzinger and M. Fuchs. Goal Oriented Equational Theorem Proving Using Team Work. In *Proc. of the KI94, Saarbrücken*, number 861 in LNAI, pages 343–354. Springer, 1994.
7. J. Denzinger and M. Kronenburg. Planning for Distributed Theorem Proving: The Teamwork Approach. In *Proc. 20^{th} KI, Dresden*, number 1137 in LNAI, pages 43–56. Springer, 1996.
8. J. Denzinger and S. Schulz. Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System. Seki-Report SR-94-05, Universität Kaiserslautern, 1994.
9. J. Denzinger and S. Schulz. Learning Domain Knowledge to Improve Theorem Proving. In M.A. McRobbie and J.K. Slaney, editors, *Proc. of the 13^{th} CADE, New Brunswick*, number 1104 in LNAI, pages 62–76. Springer, 1996.
10. J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, (Special Issue on Parallel Symbolic Computation), to appear, 1996.
11. M. Fuchs. Learning Proof Heuristics by Adapting Parameters. In *Proc. of the 12^{th} International Workshop on Machine Learning*, pages 235–243, San Mateo, CA, 1995. Morgan Kaufmann.
12. T. Hillenbrand, A. Buch, R. Vogt, and B. Löchner. Waldmeister: High Performance Equational Deduction. In Journal of Automated Reasoning, This issue, 1997.