Flexible Proof-Replay with Heuristics

Marc Fuchs
Institut für Informatik
TU München
80290 München
Germany
e-mail: fuchsm@informatik.tu-muenchen.de

April 7, 1997

Abstract

We present a general framework for developing search heuristics for automated theorem provers. This framework allows for the construction of heuristics that are on the one hand able to replay (parts of) a given proof found in the past but are on the other hand flexible enough to deviate from the given proof path in order to solve similar proof problems. We substantiate the abstract framework by the presentation of three distinct techniques for learning appropriate search heuristics based on so-called features. We demonstrate the usefulness of these techniques in the area of equational deduction. Comparisons with the renowned theorem prover OTTER validate the applicability and strength of our approach.

1 Introduction

Automatic deduction in general and particularly automated theorem proving can be interpreted as a search problem that entails huge search spaces. The problem in automated theorem proving is trying to decide whether or not a given (proof) goal λ_{th} is a logical consequence of a given set of axioms Ax. Although this problem is not decidable in general it is possible in many cases to construct algorithms able to recognize each valid goal. These algorithms are usually represented by an inference system where only weak *fairness conditions* have to be satisfied in order to guarantee completeness. But this entails an indeterminism that can only be tackled with heuristics since usually no a priori knowledge exists to help to solve the conflicts caused by the fact that often a lot of inference rules are applicable.

The difficulties in overcoming this indeterminism are the main reason why automated theorem provers are inferior to human mathematicians when trying to prove difficult problems. Thus improvements in automated reasoning systems are needed in order to allow for a more "intelligent" search thus avoiding a complete and exhaustive enumeration of all theorems of a given theory.

A possible method to increase the power of a theorem prover is the use of solved examples to solve new (and harder) problems. Theorem proving by analogy tries to find a proof for a given *target problem* $\wp_T$ by using the proof of a similar *source problem* $\wp_S$. Usually, constructive approaches are applied in order to try to transform inference steps of the source to the target using analogous matches (cp. [14]). But this application of analogy is quite problematic in the area of automated deduction. This is due to the fact that a direct transformation of proof steps usually fails because of differences in the problem descriptions. Therefore, sophisticated patching strategies are necessary in order to "fill the gaps" which are not covered by the analogical reasoning.

Therefore, we will focus in this paper on the heuristical use of proofs. We will discuss the advantages of such an approach (cp. [10]), and we will present a general framework for reusing proofs by heuristic means. This framework allows for the construction of heuristics which perform a proof-replay of a given source proof. Moreover, the heuristics offer the flexibility to deviate from the given proof path which is needed in order to solve target problems that are similar to the given source problem.

We demonstrate the applicability of our general framework by the construction of three search-guiding heuristics based on "features". Features describe structural properties of the objects that are manipulated by a theorem prover. By mapping them to numbers they allow for an abstraction of the concrete objects. The choice of features was made according to the good results obtained by using them in different ways before (e.g. [15], [10]). Our methods using features are based on the *nearest-neighbour rule* ([5]) that is refined in various ways in order to offer sufficient support to guide the search for a new target.

We evaluate the performance of our approach with several experiments in the area of equational reasoning. We have chosen the domains of Robbins algebra and group theory for performing experiments with the equational prover DISCOUNT [1]. We show that we can increase the performance of DISCOUNT in a domain (Robbins algebra) in which its performance has previously been rather weak. Furthermore, we can even improve the results of DISCOUNT in the area of group theory although fairly sophisticated heuristics are already in use (cp. [7]).

Section 2 gives an overview of automated deduction with synthetic calculi we will concentrate on in the sequel. Section 3 shows the principles of our approach to reusing proofs. Next, section 4 describes concrete search heuristics based on features. In section 5 we describe the fundamentals of equational deduction and discuss some problems regarding the use of features in this area. Section 6 presents an excerpt from our experimental results, and finally section 7 closes the paper with some conclusions and an overview of future work.

2 Automated Deduction with Synthetic Calculi

Problems of automated deduction can in general be specified as follows: From a given set Ax of facts[1] (*axioms*), we must decide whether or not a further fact λ_{th} (*goal*) is a logic consequence of Ax. A proof problem is hence given as $\wp = (Ax, \lambda_{th})$.

There are essentially two methods to tackle a proof problem $\wp$. *Analytic calculi* attempt to recursively break down and transform a goal into sub-goals that can finally be proven immediately with the axioms. *Synthetic calculi* go the other way by continuously producing logic consequences of Ax until a success inference can be applied. We shall concentrate on synthetic calculi here.

A common principle for solving proof problems algorithmically with a synthetic calculus is employed by most automated deduction systems based, e.g. on resolution ([4]) or the Knuth–Bendix completion procedure ([3]). Essentially, an automated deduction system maintains a set F^P of so-called *potential facts* from which it selects and removes one fact λ at a time. λ is put into the set F^A of *activated facts*, or discarded if it is subsumed by an already existing activated fact $\lambda' \in F^A$ (*forward subsumption*). Activated facts, unlike potential facts, are allowed to produce new facts via the application of given inference rules. The inferred new facts are put into F^P. At the beginning, $F^A = \emptyset$ and $F^P = Ax$. The indeterministic selection or *activation step* is realized by heuristic means. To this end, a search-guiding heuristic $\mathcal{H}$ associates a natural number $\mathcal{H}(\lambda) \in \mathbb{N}$ with each $\lambda \in F^P$. Subsequently, that $\lambda \in F^P$ with the smallest weight $\mathcal{H}(\lambda)$ is selected. Ties are broken according to the FIFO-strategy ("*first in–first out*"). This way we do not control every inference but restrict ourselves to a special choice

[1] A fact may be a clause, equation, or a general first or higher-order formula.

point (activation step). Experience has shown that this is a viable approach since controlling each inference step is quite expensive. Furthermore, good heuristics for selecting the next activated fact usually lead to proofs in a negligible amount of time despite the fact that time is spent on unnecessary inferences.

There are several possibilities for defining the proof obtained by the deduction process. The first natural definition of a proof is based on the inference chain performed by the prover resulting in the application of a success inference. Since this inference chain usually contains a lot of unnecessary steps we can extract an inference chain $\mathcal{I}$ by deleting the steps not contributing to the success inference. This inference chain $\mathcal{I}$ is called a *proof.*

Another possibility for defining a proof according to our control strategy is to consider the facts activated by $\mathcal{H}$ during the deduction process. If $\mathcal{H}$ succeeds in proving $\wp$, we obtain a *search protocol* $\mathcal{S} \equiv \lambda_1; \ldots; \lambda_n$ $(n \geq 1)$ which contains the activated facts. λ_n concludes the proof. By tracing back the application of inference rules starting with λ_n, all those facts λ_i of $\mathcal{S}$ which actually contributed to deducing λ_n can be identified. They are collected together with λ_n in the set P of *positive facts*. The complementary set $N = \{\lambda_1, \ldots, \lambda_n\} \setminus P$ is the set of *negative facts*. By omitting all $\lambda \in N$ from $\mathcal{S}$ we obtain a sequence $\mathcal{P} \equiv \lambda_1^+; \ldots; \lambda_m^+$, $\lambda_m^+ \equiv \lambda_n$ and $m \leq n$, which is a "stripped-down" version of $\mathcal{S}$ which contains no activation steps that are irrelevant regarding the deduction of λ_m^+ (λ_n). Note that $\mathcal{S}$, $\mathcal{P}$, P and N depend on $\mathcal{H}$ and $\wp$, but we shall make this dependency explicit only if it is necessary to avoid confusion. Although $\mathcal{P}$ may be considered as merely providing instructions on how to attain a proof of $\wp$ by traversing the search space efficiently it is nevertheless a correct chain of reasoning; its components either are axioms or were derived via inference rules involving facts occurring earlier in $\mathcal{P}$. We therefore also call $\mathcal{P}$ a *proof*. In the following, we denote proofs represented by inference chains by $\mathcal{I}$ whereas proofs represented by activation chains are denoted by $\mathcal{P}$.

When solving a problem $\wp$ we obtain a so-called *proof experience* $\mathcal{E} = (\mathcal{H}, \mathcal{S})$. The quality of a heuristic can be measured by using the following notion of redundancy: The redundancy R of a search $\mathcal{S}$ performed with heuristic $\mathcal{H}$ is defined by $R = \frac{|N|}{|P|+|N|}$. An optimal heuristic for a problem $\wp$ $(R = 0)$ only activates facts which contribute directly to the proof. In practice, we can see that conventional heuristics even if they are successful in proving a goal show a rather high degree of redundancy. Often, more difficult problems are often out of reach of an automated theorem prover because of huge search spaces and often poorly performing heuristic guiding. Altogether, there exists a great demand for improvements in search-guiding heuristics.

3 A Heuristical Framework for Proof-Replay

Although there is wide agreement that automated theorem proving systems could be improved by using past proof experience the reusing of proofs is quite complicated. This is due to the fact that in the area of automated deduction "small differences between problems usually do *not* result in small differences of their solutions". In order to overcome this problem we propose a heuristical usage of past proof experience, more exactly of one given proof, as discussed in [10] in more detail.

Usual methods of applying analogy try to *compute* a proof (inference chain) $\mathcal{I}_T$ of a target problem $\wp_T$ with the help of analogous matches from a solved source problem $\wp_S$ to $\wp_T$. But this normally necessitates sophisticated patching strategies if a direct transformation of inference steps from the source to the target fails because of the differences in their problem descriptions (cp. [12]). Therefore, such strategies are normally only well suited if a high degree of similarity between a source and a target problem exists. However, similarity between proof problems is difficult to determine a priori.

Hence, we do not try to compute a proof $\mathcal{I}_T$. We still search for it with a conventional deduction system that uses a source proof $\mathcal{I}_S$ delivered by a heuristic $\mathcal{H}$ and tries to prove $\wp_T$ by using a heuristic based on $\mathcal{H}$ that is further incorporated with information on $\mathcal{I}_S$. Such a heuristic is well suited for solving problems similar to the source problem $\wp_S$ because we can achieve a suitable compromise between flexibility (allowing for small deviations from $\mathcal{I}_S$) and specialization (avoiding useless inferences for the target). The flexibility stems from the "original" heuristic $\mathcal{H}$ which should usually be general enough to work quite well for a large set of problems. We achieve specialization by the incorporation of information on the similar source proof.

In order to realize such reusing of proofs by heuristic means it is sensible to use our second notion of a proof. This is because we assume that not every inference step is controllable but only the activation step taken according to a heuristic weighting. A heuristic specialized in the source proof can be constructed by weighting all facts $\lambda^+ \in P_S$ which are positive w.r.t. the source proof $\mathcal{P}_S$ smaller than the minimal weighting of a negative fact $\lambda \in N_S$. Furthermore, we should take care that such a heuristic does not give facts $\lambda \notin P_S \cup N_S$ a heuristic weight smaller than the maximal weight of positive facts. This could result in a completely different proof run compared with the proof obtained by $\mathcal{H}$. In particular, it is not guaranteed that such a heuristic will reach the goal (in an acceptable time limit). The following technique of *flexible proof-specialization* of a heuristic keeps these requirements in mind and offers a general framework for constructing search-guiding heuristics using information on a specific source proof. We choose the approach to penalize facts $\lambda \notin P_S$ with a penalty function ω. Thus, we get the following definition of a ω-specialization of a heuristic $\mathcal{H}$.

Definition 3.1 *Let $\wp$ be a solved proof problem with proof experience $\mathcal{E}$ and proof $\mathcal{P}$ given over a set of facts $\mathcal{O}$. Furthermore, let $\mathcal{S}$ be the sequence of facts activated by heuristic $\mathcal{H}$ while searching for $\mathcal{P}$. P denotes the set of positive facts. Let $\omega : \mathcal{O} \to I\!N$ be a (penalty) function. We call the search heuristic $\mathcal{H}_\omega$ defined by $\mathcal{H}_\omega(\lambda) = \mathcal{H}(\lambda) + \omega(\lambda)$ an ω-specialization of $\mathcal{H}$ (w.r.t. $\mathcal{E}$) if $\omega(\lambda^+) = 0 \; \forall \lambda^+ \in P$.*

ω is a function used to estimate whether a fact to be judged by some heuristic $\mathcal{H}_\omega$ contributes to a proof or not. It is the main task of ω to allow an ordinal weighting on the facts by $\omega(\lambda_1) > \omega(\lambda_2)$ iff λ_2 has a higher probability of usefulness for a proof than λ_1. In order to construct heuristics specialized in a proof also an appropriate cardinal weighting is necessary. The degree of specialization of a heuristic $\mathcal{H}_\omega$ in a proof $\mathcal{P}$ heavily depends on the values ω assigns to negative facts. We represent the degree of specialization achieved when using ω by the notion of *e-consistency*:

Definition 3.2 *Let $\wp$ be a source problem with proof experience $\mathcal{E} = (\mathcal{H}, \mathcal{S})$. Let $\mathcal{P}$ be the proof extractable from $\mathcal{E}$. P and N denote the sets of positive and negative facts, respectively. Let $e \in [0; 100]$ be a constant. Furthermore, let $\mathcal{H}_\omega^+$ be the maximal weight of a positive fact w.r.t. $\mathcal{H}_\omega$ ($\mathcal{H}_\omega^+ = \max\{\mathcal{H}_\omega(\lambda^+) : \lambda^+ \in P\}$). An ω-specialization $\mathcal{H}_\omega$ is said to be e-consistent with $\mathcal{E}$ if $|\{\lambda : \lambda \in N \wedge \mathcal{H}_\omega(\lambda) > \mathcal{H}_\omega^+\}| \geq \frac{e}{100} \cdot |N|$.*

Thus, an ω-specialization $\mathcal{H}_\omega$ being 100-consistent with $\mathcal{E}$ (associated with problem $\wp$) excludes all of the negative facts from the search and is able to obtain the proof $\mathcal{P}$ without any redundancy. Smaller consistency-values with $\mathcal{E}$ increase the overhead that still remains when using $\mathcal{H}_\omega$ for proving $\wp$. The following theorem illustrates the effect of the usage of an ω-specialization $\mathcal{H}_\omega$ being e-consistent with proof experience $\mathcal{E}$.

Theorem 3.1 *Let $\wp$, $\mathcal{E}$, $\mathcal{P}$, P, and N be as in the previous definition. Let ω be a penalty function such that $\mathcal{H}_\omega$ is an ω-specialization and $\mathcal{H}_\omega$ is e-consistent with $\mathcal{E}$ for some $e \in [0; 100]$. Then it is true that a deduction process using $\mathcal{H}_\omega$ proves $\wp$ with proof $\mathcal{P}'$ equal to $\mathcal{P}$. Furthermore, the set of positive facts P' is given by $P' = P$. For the redundancy R' of the search $\mathcal{S}'$ performed with $\mathcal{H}_\omega$ it holds $R' \leq \frac{\eta \cdot |N|}{|P| + \eta \cdot |N|}$ whereby $\eta = \frac{100-e}{100}$. Particularly, it holds $R' < R$ if $e > 0$ where R denotes the redundancy of the search obtained using $\mathcal{H}$.*

Note that it can be sensible to allow for some redundancy (use $\mathcal{H}_\omega$ that is not 100-consistent with $\mathcal{E}$) in order to avoid "over-specialization". Over-specialization arises mainly because of our inability to develop a penalty function ω which judges all facts appropriate while searching for a proof of a new target (cf. the following section). The "intelligence" or practical success of an ω-specialization (e-consistent with a proof experience belonging to a source problem $\wp_S$) in solving target problems $\wp_T$ similar to $\wp_S$ is heavily influenced by the realization of ω. A naive realization, e.g. $\omega(\lambda) = pen\_fac \cdot (1 - \chi_{P_S}(\lambda))$ (χ_{P_S} denotes the characteristic function of the positive facts w.r.t. the source proof) allows a solution of the source problem to be found more quickly (with increasing pen_fac). But it offers no real support for the solution of targets which are similar but not equal to $\wp_S$. The reason for this is that ω only favors facts *syntactically* equal to positive facts of the source. A better solution would be to favour facts that share some properties with positive facts and hence are *structurally* equal to them. Hence, we are interested in the detection of more general tendencies what positive or negative really means.

Due to positive experience obtained with so-called features in the past (e.g., [15], [10]) we want to instantiate our abstract framework described above with some concrete realizations of ω based on features. In the following section we introduce the feature concept, and then we discuss three different techniques which can be used to develop penalty functions based on features.

4 Heuristical Proof-Specialization with Features

As discussed in the previous section we want to realize "intelligent" penalty functions by using features. Features describe structural properties of facts. We want to restrict ourselves to represent such properties by functions $f : \mathcal{O} \rightarrow \mathbb{Z}$ where $\mathcal{O}$ is the set of facts. We call f a *feature*, the value $f(\lambda)$ is called the *feature value* of λ w.r.t. f. In order to allow for a better distinction between positive and negative facts and to detect properties typical for useful and useless facts, it is sensible to use several (distinct) features $f_1, \ldots, f_n$. Hence, we represent a fact λ by its *feature value vector* $(f_1(\lambda), \ldots, f_n(\lambda))$. We assume that a fixed sequence of features $f_1, \ldots, f_n$, is given and we define $FV(\lambda) = (f_1(\lambda), \ldots, f_n(\lambda))$.

In the following we are looking for penalty functions ω based on features that estimate whether a fact might contribute to a proof of a target problem $\wp_T$ or not. As usual let $\wp_S$ be a solved source problem and $\mathcal{E} = (\mathcal{H}, \mathcal{S})$ be a proof experience. P_S, N_S denote the sets of positive and negative facts, respectively. A common principle for classifying an element in artificial intelligence is to use so-called nearest-neighbour techniques (cp. [5]). The nearest-neighbour rule (NNR) in our context allows for a fact λ to take on the "class" (positive, negative) of a "nearest" fact $\lambda' \in P_S \cup N_S$ according to some predefined distance measure d (based on features) as the class of λ. But since we want to use a function ω not only for classifying elements, but also for an estimation of probabilities, we modify this technique by explicitly using distances to the nearest positive and negative facts. In the following, we present three different techniques stemming from the NNR that are based on the explicit use of distance measures in order to rate facts. We start with a simple distance measure (constructed by hand)

and study the effects arising from the use of more and more information on the source proof which is used to learn appropriate distance measures automatically.

A Penalty Function Based on the Euclidean Distance

Our first technique to construct a penalty function ω_1 uses a distance measure $d : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$. $d(\lambda_1, \lambda_2)$ is based on the Euclidean distance of the feature value vectors of the facts λ_1 and λ_2. The Euclidean distance is a standard distance measure in the area of metrically represented data. But before we explain our technical realization of d and ω_1 in detail we should at first clarify for which objects a distance is important and how this distance should be interpreted. Therefore, let d be a given distance measure defined over facts. If we assume a target $\wp_T$ similar to a source problem $\wp_S$ it is sensible to consider facts λ similar to at least one positive fact "at the feature level" to be positive but only if no negative facts exist whose feature representations are similar to λ.

We restrict ourselves to the consideration of one positive fact, namely the fact $\lambda^+ \in P_S$ which has the smallest difference d w.r.t. λ. We do so because we cannot assume that all positive facts contain similar or even equal feature value vectors. Therefore, we would risk giving (deducible) positive facts $\lambda^+ \in P_S$ a penalty although they are the most likely to contribute to a proof of the target problem $\wp_T$. Furthermore, we do not take all negative facts into consideration. Experience shows that it is not sensible to regard the distance of negative facts which are very dissimilar to λ. This way we only use the distances of the k ($k \in \mathbb{N}$, $k > 0$) nearest negative facts w.r.t. a fact λ whose usefulness we want to estimate. Let $d_1, \ldots, d_k$ be these distances. Using a constant factor $f_{pen} \in \mathbb{R}$ our realization of the function ω_1 is as follows:

$$\omega_1(\lambda) = \left\lfloor f_{pen} \cdot \frac{\min\{d(\lambda, \lambda^+) : \lambda^+ \in P_S\}}{1 + \sum_{j=1}^{k}(2^{-j} \cdot d_j)} \right\rfloor$$

The factor f_{pen} controls the redundancy still remaining if we try to solve $\wp_S$ with (ω_1-specialization) $\mathcal{H}_{\omega_1}$ within the limits given by the ability of the features to distinguish the negative from the positive facts. This means that an increase of f_{pen} can at most result in an e_{max}-consistency with proof experience $\mathcal{E}$ where e_{max} is the percentage of negative facts containing a different feature value vector from all positive facts. Note that it is sensible to allow for some redundancy (even if possibly $e_{max} = 100$) since it could be the case that a fact λ is needed although it is a rather high distance away from positive facts. If we do not allow for some redundancy this possibly causes a rather large factor f_{pen}. But this can result in a weight $\mathcal{H}_{\omega_1}(\lambda)$ that makes an activation of λ impossible within an acceptable amount of time. Therefore, we try to find a compromise between a sufficient degree of specialization and flexibility.

In order to find an appropriate setting of f_{pen} we consider the value e_{max} that is the maximal value e that allows for an e-consistency of $\mathcal{H}_{\omega_1}$ w.r.t. $\mathcal{E}$. If this value e_{max} is rather low it is sensible to fully exploit the given potential for eliminating negative facts from the search for the source proof. But if e_{max} is rather high it could be better to allow for more redundancy. A possible compromise is to use $e_{fs} = \min\{e_{max}, e_f\}$ for a given constant $e_f < 100$. In our experiments settings of e_f in the range of $[70; 90]$ reached the best results. Utilizing e_{fs} a possible method is to set f_{pen} as the minimal factor that allows for an ω_1-specialization of $\mathcal{H}$ that is e_{fs}-consistent with $\mathcal{E}$.

Finally, we want to discuss in which way an appropriate distance measure d can be defined. As mentioned above we want to use the Euclidean distance to realize d. But a direct use of the Euclidean distance at the feature value vectors could cause some problems because distinct features could have different ranges. Different ranges can lead to *implicit weightings* of features. A feature with a rather large range (e.g.,[0;100]) implicitly has more influence for judging a fact than a feature with a smaller range (e.g.,[0;1]). Hence, it is sensible to *standardize* the feature value vectors w.r.t. the data provided by the source proof. We use the function $\eta : \mathbb{Z}^n \to \mathbb{Z}^n$ defined by $\eta(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where $y_i = \frac{x_i - \mu_i}{\sigma_i}$, σ_i and μ_i denote the variance and the average of the set $\{f_i(\lambda) : \lambda \in P_S \cup N_S\}$, respectively. This transformation yields to $M_i = 0$ and $S_i = 1$ whereby S_i and M_i denote the variance and the average of the set $\{\eta(FV(\lambda))_i : \lambda \in P_S \cup N_S\}$, respectively. Altogether, a distance measure avoiding implicit weightings of features can be defined by $d(\lambda_1, \lambda_2) = (\sum_{i=1}^{n}(z_{1i} - z_{2i})^2)^{\frac{1}{2}}$ with $z_j = \eta(FV(\lambda_j))$, $j = 1, 2$.

Learning a Weighted Distance Measure

Since we want to estimate the usability of a fact λ by using its feature value vector the technique previously used is surely justified. But our technical realization can be furthermore improved. Indeed, we have developed a measure without implicit weightings of features but we do not explicitly consider the importance of certain features according to their ability to distinguish positive from negative facts. Hence, we should try to use an *explicit weighting* of features w.r.t. their assumed importance. This way we use the distance measure $d_{a_1,\ldots,a_n} : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ parameterized by $(a_1, \ldots, a_n) \in \mathbb{N}^n$ when using features $f_1, \ldots, f_n$. It is defined by $d_{a_1,\ldots,a_n}(\lambda_1, \lambda_2) = (\sum_{i=1}^{n} a_i(x_i - y_i)^2)^{\frac{1}{2}}$ if $FV(\lambda_1) = (x_1, \ldots, x_n)$ and $FV(\lambda_2) = (y_1, \ldots, y_n)$. Important features can get a higher influence on the distance value than less important features by giving them a higher coefficient. Furthermore, features f_i associated with coefficient a_i with $a_i = 0$ can be neglected so as to contribute to a higher degree of abstraction compared with the method described above. The use of $d_{a_1,\ldots,a_n}$ also offers some advantages w.r.t. the computation time when compared with d. Since we can define the coefficients in such a way that a small (Euclidean) distance to negative facts (which are distinguishable from positive facts) results in a high distance, no explicit consideration of negative facts is needed. Altogether, a possible realization of a penalty function ω_2 is

$$\omega_2(\lambda) = \lfloor \min\{d_{a_1,\ldots,a_n}(\lambda, \lambda^+) : \lambda^+ \in P_S\} \rfloor$$

Using this realization the coefficients control the importance of distinct features as well as the remaining redundancy when proving the source problem $\wp_S$. Finally, we have to discuss how an appropriate setting of the coefficients $a_1, \ldots, a_n$ can be achieved. It is clear that this setting should not be chosen by a human reasoner but has to be learned automatically. Due to the lack of space we cannot explain our method in detail. But in order to give a rough idea how to compute coefficients we should clarify what the "importance" of a feature means. A feature f is said to be important if it is able to distinguish the negative from the positive facts w.r.t. the feature values of f. Hence, if a small increase in the coefficient associated with a feature f results in a rather high increase of the values $\omega_2(\lambda)$ of a high number of negative facts f can be considered to be quite important. For a more detailed description of our method we refer to [10] where a similar technique is used to compute coefficients for a heuristic that weights distances between feature values of facts and so-called permissible feature values. It should be mentioned that similar to our first method the algorithm for computing coefficients uses a parameter e_f. This constant is responsible for controlling the percentage of negative facts that disappear from the search for the source proof which is carried out with $\mathcal{H}_{\omega_2}$ as described before.

Abstraction from Positive Feature Value Vectors

Similar to the previous technique we are interested in recognizing relevant features and in the design of a penalty function ω_3 that rates the distance of a fact λ to the nearest positive fact $\lambda^+ \in P_S$ w.r.t. the relevant features. But compared with the technique described above we even want to pay more attention to the source proof in order to learn an appropriate distance measure. This way we try to find out for every positive fact which deviations from its feature value vector can cause problems (making it similar to negative facts) and which are harmless. The central idea is now only to penalize deviations from "positive feature value vectors" that are quite similar to the feature values of negative facts. This idea is similar to the concepts applied to find the coefficients $a_1, \ldots, a_n$ of our second method but here we want to consider each positive fact separately.

This way we abstract from the actual positive facts and only represent each of them by one "positive (contiguous) area" containing the positive fact. The facts contained in these areas all are considered to be positive. As implied above the construction of these positive areas is influenced by the negative facts. We are interested in a rather high abstraction from a positive fact that is consistent with the negative facts, i.e. no negative fact should be contained in a positive area. The function ω_3 is intended to rate the distance of a fact to the nearest positive area instead of the distance to the nearest positive fact.

In order to substantiate these abstract principles we should at first choose an appropriate structure of a positive area. We have chosen the approach to represent a positive area A_{λ^+} associated with a positive fact λ^+ in the following way (assuming features $f_1, \ldots, f_n$):

$$A_{\lambda^+} = A_{\lambda^+, a_1, \ldots, a_n, \epsilon} = \{\lambda \in \mathcal{O} : \sum_{i=1}^{n} a_i \cdot (f_i(\lambda^+) - f_i(\lambda))^2 \leq \epsilon^2\}$$ [2]

It holds $a_i \in \mathbb{R}$, $a_i \geq 0$, $1 \leq i \leq n$, and $\epsilon \in \mathbb{R}$, $\epsilon \geq 0$. If $a_i > 0$, $\forall i \in \{1, \ldots, n\}$, then A_{λ^+} represents all facts whose feature value vectors are placed in the n-dimensional ellipse that is parameterized by $a_1, \ldots, a_n$, ϵ and its center $FV(\lambda^+)$. Note, however, that it is explicitly allowed to use $a_i = 0$ for a feature f_i. In this way we can completely abstract from some features in order to measure distances to certain positive facts.

In order to define a penalty function ω_3 we assume the existence of a positive area A_{λ^+} for each positive fact $\lambda^+ \in P_S$. A possible definition of ω_3 is

$$\omega_3(\lambda) = \lfloor \min\{d_A(\lambda, A_{\lambda^+}) : \lambda^+ \in P_S\} \rfloor$$

d_A is a distance measure based on features where $d_A(\lambda^+, A_{\lambda^+}) = 0 \ \forall \lambda^+ \in P_S$. Altogether, our definition of ω_3 allows for proof specializations of given source proofs. In the sequel we shall show how positive areas can be learned and how to construct an appropriate distance measure d_A.

At first we want to take a closer look at the construction of positive areas. A positive area A_{λ^+}, $\lambda^+ \in P_S$, is defined by the parameters $a_1, \ldots, a_n$ and ϵ. Thus we are looking for settings of these parameters. A sensible method is to displace the learning of A_{λ^+} to the maximization of a function $\varphi_{\lambda^+} = \varphi_{\lambda^+}(a_1, \ldots, a_n, \epsilon)$. The function value of φ_{λ^+} should measure the degree of abstraction from λ^+ that is consistent with the negative facts. Since the size of the set A_{λ^+} is usually infinite we have to use another value in order to measure the degree of abstraction from λ^+ obtained by using A_{λ^+}. Thus we use a value approximately proportional to the size of the n-dimensional ellipse defined by $FV(\lambda^+)$, $a_1, \ldots, a_n$ and ϵ. We define

$$\varphi_{\lambda^+}(a_1, \ldots, a_n, \epsilon) = \begin{cases} \frac{\epsilon^n}{\prod_{i=1}^{n}(\sqrt{a_i}+1)} & ; A_{\lambda^+, a_1, \ldots, a_n, \epsilon} \cap N_S = \emptyset \\ 0 & ; otherwise \end{cases}$$

$\varphi_{\lambda^+}(a_1, \ldots, a_n, \epsilon)$ is large if the corresponding area $A_{\lambda^+, a_1, \ldots, a_n, \epsilon}$ is large and contains only positive facts. We have used a simulated annealing algorithm SA

[2] Naturally, the coefficients $a_1, \ldots, a_n, \epsilon$ belong to a certain $\lambda^+ \in P_S$ but in the following we will not make this dependency explicit.

in order to learn coefficients separately for each positive fact. This algorithm starts to learn $a_1, \ldots, a_n$, ϵ for a fixed positive fact λ^+ with the initial setting $a_1 = \ldots = a_n = 1$. ϵ is set to the greatest value that satisfies that no negative fact is contained in A_{λ^+}. We restricted the solution space of the following maximization process performed by the SA to a discrete universe (a n+1-dimensional regular grid) that contains the initial setting.

Finally, we should clarify how the distance measure d_A is defined. We rate the distance between a fact λ and an area A_{λ^+} with the help of the Euclidean distance between "$FV(\lambda)$ and the ellipses defined by A_{λ^+}". We do not want to give an exact definition here. We only want to mention that similar to the previous method the learning process of the distance measure d_A can be controlled by a parameter e_f which limits the degree of e-consistency reached by the ω_3-specialization $\mathcal{H}_{\omega_3}$ w.r.t. the proof experience $\mathcal{E}$.

5 Features in the Area of Equational Deduction

In this section we give at first a brief introduction into the area of equational reasoning. We have chosen this area to evaluate our concepts with some experiments (mainly because of the existence of an implementation). Since the techniques presented before heavily depend on the quality of the features we discuss afterwards how features should be constructed within the context of equational reasoning.

5.1 Unfailing Completion

Equational reasoning deals with the following problem: Does it hold $Ax \models u = v$ for a given set $Ax = \{s_i = t_i : 1 \leq i \leq n, n \in \mathbb{N}\}$ of equations (of terms over a fixed signature sig) and a goal $u = v$?

The completion method by Knuth and Bendix ([11], extended to unfailing completion, [3]) has proven to be quite successful for solving this problem. This method is a typical example of a synthetic or bottom-up calculus. In the following, we assume the reader to be familiar with rewriting and completion techniques. For an overview see [8] or [2].

As usual, a signature sig is a pair (F, τ) with a set of operators F and a function $\tau : F \to \mathbb{N}$ that returns the arity of an operator. The set of terms $T(F, V)$ to a signature and a set V of variables is defined as the minimal set fulfilling $x \in T(F, V)$, if $x \in V$ and for $f \in F$ with $\tau(f) = n$ and terms $t_1, ..., t_n$ $f(t_1, ..., t_n) \in T(F, V)$.

Basically, the inference rules of a synthetic prover can be divided into two classes: *extension* rules and *contraction* rules. Completion uses the extension rule *critical-pair-generation* and the contraction rule *reduction*. A basis for the completion procedure is a so-called reduction ordering $\succ$ that is used to restrict the applicability of the inference rules and avoids cycles. In the following definitions, the right and left hand sides of equations may be exchanged.

A critical pair $s = t$ to two equations $l_1 = r_1$ and $l_2 = r_2$ is defined as the equation $\sigma(l_1)[p \leftarrow \sigma(r_2)] = \sigma(r_1)$, if p is a position in l_1 (and $l_1/p \notin V$), such that σ is the mgu of l_1/p and l_2 and if $\sigma(l_1)[p \leftarrow \sigma(r_2)] \not\succ \sigma(l_1)$ and $\sigma(r_1) \not\succ \sigma(l_1)$. The resulting equation is a valid consequence of the two parent equations.

A reduction of an equation $s = t$ by an equation $l = r$ replaces it by $s' = t$. A reduction only may be performed if there is a position p in s such that there is a match μ from l to s/p, i.e. $\mu(l) \equiv s/p$, and $\mu(l) \succ \mu(r)$. Then s' is defined by $s' \equiv s[p \leftarrow \mu(r)]$. If there is no equation that can reduce a term in an equation, then the term (and the equation) are in *normal form*. The normalization of a term (or equation) is always a finite process.

To solve problems algorithmically using unfailing completion an algorithm based on the principles described in section 2 is applicable. One uses a set CP corresponding to the potential facts, and two sets R and E describing the active facts. R contains the set of processed equations that are orientable with $\succ$. E contains the processed equations that are not orientable w.r.t. $\succ$. As described before it is possible to realize the completion procedure in such a way that the only indeterminism that remains is which facts should be selected from the set CP in order to perform further inferences.

5.2 Features for Equations

In order to construct features for equations we should take care that the equational relation is symmetric. Hence, two equations $u = v$ and $v = u$ that are equivalent during the completion process should obtain equal feature value vectors. But although the equality relation is symmetric we should not neglect the fact that equations can become oriented equations (rules) during the proof process. This is fairly important since we should not, e.g., make two rules $l \to r$ and $l' \to r'$ equal (or similar) at the abstraction level of feature value vectors even if l and r' as well as r and l' are quite similar. Considering the role the rules can play in the proof process (especially if l and l' as well as r and r' are quite dissimilar) they should not contain similar feature values. Thus a feature function should on the one hand ignore the physical ordering of the equations produced by the deduction system. But on the other hand a feature should be aware of the existence of the ordering used in the deduction process and take the reduction ordering into account when computing a feature value.

We choose the approach to *standardize* the equations according to the reduction ordering, and then use the standardized equations in order to compute feature values. The process of standardizing an equation is carried out by reordering the equation w.r.t. a total ordering of terms. Hence, we assume that a fixed reduction ordering $\succ$ is given. Furthermore, let $\rho_\succ$ be a relation on the given set of terms that includes $\succ$ ($\succ\subseteq \rho_\succ$). Moreover, it should hold for $u \not\equiv v$ that $\rho_\succ(u,v)$ or $\rho_\succ(v,u)$ but not both (simultaneously). Note that such a relation $\rho_\succ$ can easily be defined using $\succ$ and any total ordering defined on the set of terms used to orient equations which are not orientable by $\succ$. Using $\rho_\succ$ we define

$$Standardize_{\rho_\succ}(u=v) = \begin{cases} u=v & ; u \equiv v \ or \ \rho_\succ(u,v) \\ v=u & ; otherwise \end{cases}$$

With the help of $\rho_\succ$ and $Standardize_{\rho_\succ}$ we define standardized features $f_{\rho_\succ}$ according to a feature f by $f_{\rho_\succ}(u=v) = f(Standardize_{\rho_\succ}(u=v))$. We want to remark that the choice of the reduction ordering $\succ$ has no influence on the fact that penalty functions using features $f_{\rho_\succ}$ result in proof specializations of a given source problem. But in order to solve a similar though not equal target problem it is sensible to choose a reduction ordering $\succ'\supseteq\succ$ in order to compute the feature values. $\succ$ denotes the reduction ordering used to prove the chosen source problem. This is absolutely necessary in order to use the feature values to estimate whether facts, similar to positive facts at the feature level, can contribute to inferences similar to the positive inferences of the source proof.

Finally, we want to give a brief overview of the features which we have used to abstract from a given fact. The features are quite simple and only consider syntactical properties of the given facts. Altogether, we used 15 features independent from the given signature and 4 feature schemata applied for each function symbol. An example of a feature is the number of function symbols introduced by an equation.[3] Other features can be found, e.g. in [9]. A feature scheme is, e.g. the number of occurrences of a specific function symbol in an equation. In the following we will not further distinguish between the standardized and the "ordinary" features behind them. In the following a feature value of a feature f is always the value of f applied to an equation standardized w.r.t. a reduction ordering containing the reduction ordering used to solve the chosen source proof.

6 Experiments with Discount

We carried out several experiments with the prover Discount. Discount is a distributed prover for pure equational logic employing the Teamwork method ([6]). Besides the use of a single heuristic in order to solve a problem, the prover also allows for the use of several concurring and cooperating heuristics, a so-called *team*. We still did not use the heuristics learned from previous examples in a team.

We only present here an excerpt from our experimental results. We restrict ourselves to the domains of Robbins algebra and group theory. We report on experiments performed with the problems available in the ROB and GRP domain

[3]Note that the standardization allows us to explicitly consider a left and a right hand side of an equation when computing feature values.

of the TPTP library ([16]) v.1.2.1. More exactly we tried to solve the problems in both domains that are not solvable with conventional heuristics of DISCOUNT within the time limit of 10 minutes using a SPARCstation 10 ("hard problems"). A comparison with the renowned OTTER prover [13] demonstrates the quality of our learning approach.

Methodology: In order to use the concepts previously introduced source proofs are needed that are in a way similar to the target problems that we want to solve. In general it is a non trivial task to choose a source problem appropriate for a given target problem. But since we only apply a heuristical reuse of proofs the choice of a source problem is not as problematic as if we had chosen a "constructive" approach.

Therefore, in order to solve the 13 (universally quantified) problems of the ROB domain that are not solvable with a conventional DISCOUNT heuristic we used only one source proof, namely ROB003-1 which is the hardest problem solvable by the conventional *addWeight* heuristic. *addWeight* simply counts the number of functions and variables of an equation. Similarly, in the GRP domain (more exactly in the area of lattice ordered groups) we chose only the proof for problem GRP179-1. We can specialize the heuristic *maxWeight* in this proof. *maxWeight* computes the maximum of the weights of the left hand side and right hand side of an equation where weight corresponds to the number of functions and variables. In the area of "standard" groups DISCOUNT only fails for two problems which also remained unsolved after employing learned heuristics. Hence, in the following we only consider the 28 problems in the area of lattice ordered groups that cannot be solved with conventional DISCOUNT heuristics (within 10 minutes).

Outgoing from the proof of ROB003-1, obtained using *addWeight*, we learned three penalty functions ω_1^R, ω_2^R, and ω_3^R according to the three techniques developed before. Although one can construct 100-consistent ω_i^R-specializations of *addWeight*, $i = 1, 2, 3$, we allowed for some redundancy by using $e_f = 80$ when learning the penalty functions. This setting performed best in our experiments.

Similarly, we specialized *maxWeight* in the proof of GRP179-1. The learning process was executed using $e_f = 80$. We learned the penalty functions ω_1^G, ω_2^G, and ω_3^G.

Experimental Results: The heuristics $addWeight_{\omega_i^R}$ and $maxWeight_{\omega_i^G}$ were applied to perform experiments in the ROB and GRP domain, respectively. The orderings used to tackle the target problems are equal to the orderings used to solve their related source problems (lexicographical path orderings). Table 1 gives an overview about the results we obtained in the GRP and ROB domain. For each domain we depict the number of successful proof runs and the accumulated run time (in seconds) counting failures with 600 seconds.

Table 1: Results in the ROB and GRP domain ("hard" problems)

GRP domain	OTTER	$maxWeight_{\omega_1^G}$	$maxWeight_{\omega_2^G}$	$maxWeight_{\omega_3^G}$
successes	3	15	15	15
acc. runtime	15008	8082	7992	8412
'winner'	3	4	11	0
ROB domain	OTTER	$addWeight_{\omega_1^R}$	$addWeight_{\omega_2^R}$	$addWeight_{\omega_3^R}$
successes	6	4	4	5
acc. runtime	4426	5600	5479	4859
'winner'	5	0	0	1

All times are measured on a SPARCstation 10. The entry of the row 'winner' denotes the number of proof runs in which the learned heuristics or OTTER performed better than the other techniques.

Altogether, we can improve the performance of DISCOUNT significantly. Although OTTER still performs better than our learning approach in the area of Robbins algebra we could increase the number of successful proofs of DISCOUNT in this area from 5 to 10. Now the success rate of DISCOUNT almost reaches that from OTTER (11 problems solved). We can also improve the results of DISCOUNT in the GRP domain. Using the conventional heuristics and the learned heuristics we are now able to solve 93 of 105 problems in the area of lattice ordered groups (78 problems solved without learning). OTTER can only solve 79 problems.

Comparing our different techniques one can see that neither technique is consistently better than the other. But a general tendency is that heuristics based on ω_1 or ω_2 seem to show a higher degree of specialization. Therefore they perform fairly well in the area of lattice ordered groups where the problems are more similar than in the robbins algebra. But in some cases where more flexibility is needed (to allow for deviations from the source proof) ω_3 improves on the performance of the other heuristics by offering a higher degree of abstraction from the positive facts. Therefore, the use of this penalty functions allows for the construction of heuristics in the domain of robbins algebra that are clearly superior to the heuristics based on the other simpler learning schemes.

7 Conclusions and Future Work

We have presented a general framework for flexible proof-replay by using heuristics. We have demonstrated the performance of our method with three distinct realizations based on features. Experiments in two domains underline the applicability of our approach. Although DISCOUNT is clearly inferior to OTTER when its inference rate is considered, we nearly achieved the results of OTTER in a domain in which the conventional heuristics of DISCOUNT perform quite badly. This, together with the fact that we could solve problems in the area of lattice ordered groups that were out of range before, is a sign that our approach is well suited for the control of automated deduction systems.

Future work needs to be done in order to allow for an automatical choice of appropriate source proofs. Especially, it is reasonable to iteratively use the proofs obtained by (learned) heuristics in order to solve a set of problems. At first the problems are tackled with conventional heuristics. After that heuristics are learned that are specialized in the proofs delivered by the heuristics previously used. These heuristics are possibly able to solve some of the harder problems that are still unsolved and a new "round" of the solution process can start.

References

[1] **Avenhaus, J.; Denzinger, J.; Fuchs, Matt.:** DISCOUNT*: A system for distributed equational deduction*, Proc. 6^{th} RTA, Kaiserslautern, FRG, 1995, LNCS 914, pp. 397–402.

[2] **Avenhaus, J. ; Madlener, K.**: *Term Rewriting and Equational Reasoning*, in R.B. Banerji (ed): Formal Techniques in Artificial Intelligence, Elsevier, 1990, pp. 1–43.

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

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

[5] **Cover, T.M. ; Hart P.E.:** *Nearest Neighbor pattern classification*, IEEE, Transactions on Information Theory 13, pp. 21-27, 1967.

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

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

[8] **Dershowitz, N. ; Jouannaud, J.P.:** *Rewriting systems*, in J. van Leeuwen (Ed.): Handbook of theoretical computer science, Vol. B., Elsevier, 1990, pp. 241-320.

[9] **Fuchs, Matt.:** *Experiments in the Heuristic Use of Past Proof Experience*, SEKI-Report SR-95-10, University of Kaiserslautern, 1995, obtainable via WWW at `http://www.uni-kl.de/AG-AvenhausMadlener/fuchs.html`

[10] **Fuchs, Matt.:** *Experiments in the Heuristic Use of Past Proof Experience*, Proc. CADE 13, New Brunswick, NJ, USA, 1996.

[11] **Knuth, D.E. ; Bendix, P.B.:** *Simple Word Problems in Universal Algebra,* Computational Algebra, J. Leech, Pergamon Press, 1970, pp. 263-297.

[12] **Kolbe, T.; Walther, C.:** *Patching Proofs for Reuse*, Proc. 8^{th} ECML '95, Heraklion, Crete/Greece, 1995.

[13] **McCune, W.W.:** *OTTER 3.0 reference manual and guide*, Techn. report ANL-94/6, Argonne Natl. Laboratory, 1994.

[14] **Owen, S.:** *Analogy for automated reasoning*, Academic Press, 1990.

[15] **Suttner, C.; Ertel, W.:** *Automatic acquisition of search-guiding heuristics*, Proc. CADE 10, Kaiserslautern, FRG, 1990, LNAI 449, pp. 470–484.

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