Exploiting past proof experience*

Matthias Fuchs
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany
E-mail: fuchs@informatik.uni-kl.de

Abstract

We are going to present two methods that allow to exploit previous experience in the area of automated deduction. The first method adapts (learns) the parameters of a heuristic employed for controlling the application of inference rules in order to find a known proof with as little redundant search effort as possible. Adaptation is accomplished by a genetic algorithm. A heuristic learned that way can then be profitably used to solve similar problems. The second method attempts to re-enact a known proof in a *flexible* manner in order to solve an unknown problem whose proof is believed to lie in (close) vicinity. The experimental results obtained with an equational theorem prover show that these methods not only allow for impressive speed-ups, but also make it possible to handle problems that were out of reach before.

1 Introduction

Automated deduction is—at its lowest level—a search problem that spans huge search spaces. The general undecidability of problems connected with (automated) deduction entails an indeterminism that has to and can only be tackled with heuristics. Mostly, automated deduction is employed to prove that a given conjecture can be deduced from an also given set of axioms. In order to solve such tasks automated proving systems utilize a (fixed) set of inference rules whose applications are responsible for both indeterminism and the immense size (the "combinatorial explosion") of the search space. Despite a far superior inference rate the computer is inferior to (human) mathematicians when it comes to proving "challenging" theorems. One prominent reason for this drawback of automated proving systems is their inability to make use of past

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

experience, which is very often quite helpful or even an indispensable key to success. Therefore, it stands to reason to upgrade automated proving systems on that score.

But exploiting past proof experience fruitfully is in general neither trivial nor does it come without hazards. The main problem is that analogy in the widest sense is hard to define, to detect and to apply in the area of automated deduction. In other branches of artificial intelligence various applications of analogy have proven to be powerful tools (see, for instance, [Ca86], [Bu89]). These research areas profit from the fact that "*small changes of the problem description (usually) cause small changes of the solution*". This is definitely not true for automated deduction (proving). Consequently, we have to be very careful about making use of past proof experience in order not to stumble into a major pitfall of this kind of reuse, namely making things considerably worse compared to proving from scratch (cp. [KN93]).

In spite of these bleak prospects of success we still think that it is worthwhile equipping an automated proving system with the option to utilize experience gained in the past.

In this report we are going to propose two distinct ways to make use of past proof experience. Firstly, the parameters of a given (generic) heuristic for controlling the application of inference rules are adapted so as to find a proof $\mathcal{P}$ of an also given proof problem $\mathcal{A}$ obtained in the past with "as little redundancy as possible". (Here, redundancy refers to the exploration of paths in the search space which do not contribute to attaining this particular proof $\mathcal{P}$.) The heuristic adapted to $\mathcal{P}$ (i.e., the heuristic has *learned* $\mathcal{P}$) can then be applied to a (novel) proof problem $\mathcal{A}'$ "similar" to $\mathcal{A}$, profiting from the reduction of redundancy. An adapted heuristic *indirectly* reuses $\mathcal{P}$ by seizing the essential properties of $\mathcal{P}$ with its parameters during the learning process, which is realized by a genetic algorithm. Secondly, we propose a method that *explicitly* utilizes a proof $\mathcal{P}$ found in the past. The method proceeds by flexibly following the path through the search space corresponding to $\mathcal{P}$. Flexibility is achieved by allowing for gradual expansions of this prescribed path. This way proofs lying in (close) vicinity of $\mathcal{P}$ can be found (very) quickly.

A number of experiments will demonstrate the viability of these two methods. Our experiments will show that exploiting past proof experience can not only give rise to salient speed-ups, but can also enable a proving system to handle problems it could not cope with before. Among these problems are tasks even such respected and powerful proving systems as OTTER 3.0 ([Mc94]) and Herky ([Zh92]) have difficulties with (see in particular subsection 6.2).

The report is organized as follows. First, section 2 gives an overview of the general problematic we have to face in this research area. Sections 3 and 4 introduce the methods 'learning proof heuristics' and 'extending proofs' for utilizing proofs found in the past. Section 5 briefly describes the proving system we employed for the experiments documented in section 6. A discussion of related work in section 7 and a summary in section 8 conclude this report.

2 The general problematic

The exploitation of past proof experience generally confronts us with the following situation: We are given a problem description $\mathcal{A}' = (Ax', Th')$, the task being to find a proof of the theorem Th' based on the axioms in Ax'. In order to accomplish this we rest on a proof $\mathcal{P}$ of some similar problem $\mathcal{A} = (Ax, Th)$ found in the past. $\mathcal{A}'$ is referred to as the *target* (problem), while $\mathcal{A}$ is called the *source* (problem). In many areas of artificial intelligence reusing solutions means modifying a known solution of a source problem so that its modification provides at least a good starting point for creating a solution of the target problem, if not already being a solution itself. A well-known representative of this line of research is *case-based reasoning* ([Ko92]). Applied to automated deduction (respectively proving) the proof $\mathcal{P}$ has to be modified so as to become a proof of the target $\mathcal{A}'$. But such modifications will in general entail new proof problems caused by differences between the axioms or theorems of source and target. This suggests a more general way of proceeding: In order to prove the target—assuming a similarity between it and the source—we try to infer facts that are in some way similar to those needed for the source proof $\mathcal{P}$. This method works perfectly if there is total correspondence between source and target, i.e., both Ax and Ax' as well as Th and Th' are identical (modulo renaming symbols). In that case, 'similar' means 'identical' and the source proof merely has to be re-enacted, thus avoiding almost any form of redundancy. But this effect can also be achieved by simply storing and retrieving proofs from a database (*memory-based reasoning*). The more interesting case arises if source and target share some properties, but do not agree completely. Under these conditions, the source proof $\mathcal{P}$ can still be useful as a guideline, but we also have to permit deviations from the path traced out by $\mathcal{P}$ in order to find a proof of the target theorem. These deviations from the source proof constitute the main problematic of any form of proof reuse in general. On the one hand, we want to stay close to the source proof to prune the search space effectively, which is the true motive of reusing proof experience. On the other hand, we cannot stick rigidly to the source proof unless source and target are identical (modulo renaming). Consequently, we need a compromise between *flexibility* and *rigidity* (*inflexibility*) or, in other words, between *generality* and *specialization*. A method for exploiting past proof experience is referred to as being specialized if it allows for a significant reduction of search effort, but requires a high degree of similarity between source and target in order to be successful. A more general method, however, does not depend on the similarity of source and target to a great extent, but naturally cannot cut down on search effort quite as effectively. So, general (flexible) and specialized (rigid, inflexible) methods for utilizing past proof experience are at opposite ends of a scale dividing up such methods. In the extreme case, a general method does not assume any similarity between source and target and hence basically proceeds ignoring the source. An extremely specialized method, however, attempts to re-enact the given source proof and will fail if this is not possible or if it cannot prove the target this way. Therefore, the degree of similarity between source and target determines the appropriate method. Unfortunately, the degree of similarity is in general very difficult to determine *a priori*.

Since we assume a certain similarity between source and target the methods we are going to present in sections 3 and 4 are situated near the "specialized end" of the scale, while still offering reasonable flexibility. The experiments described in section 6 reveal that a lot of problems with a varying degree of similarity can be profitably tackled with these methods. We should like to emphasize that the success of our methods not only consists in substantial speed-ups, but also in making it possible to solve problems which could not be handled before.

3 Learning proof heuristics

Our first method for exploiting past proof experience originates from the following considerations: Almost any automated proving system disposes of a set of heuristics for controlling the application of its inference rules. Most of these heuristics are very general (flexible) in the sense of section 2, because they are successfully applicable to a wide range of problems (without falling back on source problems). Also, such heuristics usually are parameterized. Consequently, each is a perfect candidate to become more specialized by "tuning" its parameters, while still retaining a fair degree of flexibility due to its inherent generality. The aim of adaptation (learning[1]) consists in creating a heuristic that is especially tailored to the needs of a given source problem $\mathcal{A}$ and a known proof $\mathcal{P}$ (the source proof) of it. Adaptation is achieved by finding parameter configurations for a given (generic) heuristic so that it can follow the path through the search space set out by $\mathcal{P}$ without losing its way too often, thereby reducing redundancy with respect to the search for $\mathcal{P}$. The limitedness of commonly used heuristics will in general prevent any learning procedure from finding a parameter configuration that makes it possible to follow that path exactly, which stands in contrast to the demand for flexibility. Hence, an adapted heuristic should be able to cope with moderate differences between $\mathcal{A}$ and a target problem. Furthermore, if an adapted heuristic is applied to a target problem which is (supposedly) similar to the source problem $\mathcal{A}$ it learned, it does not consult the source proof $\mathcal{P}$ anymore. The "essence" of $\mathcal{P}$ has been assimilated in the course of adaptation, making any further use of $\mathcal{P}$ obsolete. Therefore, this method so to speak *indirectly* reuses proofs, and it has the advantage of not causing any kind of overhead during its application.

After this basic description of our first method, we have to address its central problematic, namely the way adaptation (resp. learning) is to be accomplished. Commonly, the parameters of a heuristic are flags or numerical values. Both types can very conveniently be represented by one or several bits. Hence, the whole set of parameters, i.e., a parameter configuration, is representable as a string of bits. This suggests the use of a genetic algorithm. The following subsection will outline its foundations as well as its application to our learning problem.

[1]We shall use the notions 'adaptation' and 'learning' synonymously.

3.1 Learning with the genetic algorithm

The genetic algorithm ([Ho75], [Da88], [Ra91])—GA for short—is an adaptive method based on principles known from general genetics and biological evolution. It is very useful for finding (near) optimal solutions to problems in many domains. It differs from other optimization techniques in that it maintains a set of individuals (usually with a fixed size) which is called a *population* or *generation*. Each of these individuals corresponds to a (sub-optimal) solution of the given problem. An individual is thus a representation of a solution that suits the GA. The GA constructs new individuals (and hence new solutions) using the best ones of its current population, replacing those considered least fit ("*survival of the fittest*"). The assessment of individuals is accomplished by the so-called *fitness function*. The construction of new individuals is achieved by applying *genetic operators* which basically reflect and simulate the processes involved in biological reproduction. The most important genetic operators are *crossover*, *mutation* (and *inversion*). An important prerequisite for the GA to be applicable is a structure of the individuals that is amenable to these genetic operators. Common is a representation as a string of bits of a fixed length n.

In our case, individuals correspond to parameter configurations of a heuristic for selecting the next (deductive) inference to be performed by an automated prover. By restricting the range of each parameter a parameter configuration can be represented by a string of bits of fixed length which is constructed by concatenating the bit representation of each parameter of the configuration. Hence no problems are raised regarding the applicability of the genetic operators. So, the major problem that remains is the design of an appropriate fitness function. The fitness function is in general very important for any GA application, because it establishes the only connection between the abstract underlying search method and the actual optimization problem. Our problem consists in finding a parameter configuration for a heuristic so that an automated prover using this (adapted) heuristic can find a proof of a given proof problem $\mathcal{A} = (Ax, Th)$ with "minimal redundancy". The guidance during the search for an optimal parameter configuration is provided by a (source) proof $\mathcal{P}$ of $\mathcal{A}$ found in the past.

Let $\mathcal{W}$ be a generic heuristic and ω an instance of $\mathcal{W}$ corresponding to a parameter configuration (an individual) we wish to assess, i.e., the parameters of $\mathcal{W}$ are set to specific values. The quality of an ω is expressed by the amount of redundancy ω would produce if a proof of $\mathcal{A}$ guided by ω were attempted. So, the ideal and most exact way to assess an ω consists in proving or attempting to prove that Th follows from Ax using ω and subsequently analyzing statistical data collected during the search. But clearly enough this is impractical. The fitness function has to be applied to each (new) individual during each cycle[2], and hence a fitness function that needs at least a couple of seconds (which has to be considered the *lower* limit to obtain expressive statistical data during a proof attempt) for rating just one individual is untenable.

The following considerations offer an efficient alternative. Most redundancies during a search for $\mathcal{P}$ are caused by inferring facts which do not contribute to finding $\mathcal{P}$. The

[2]One cycle of the GA comprises the rating of all individuals, selecting the best and replacing the rest with offspring of the best.

source proof $\mathcal{P}$ corresponds to a particular path in the search space. With the help of $\mathcal{P}$ all (potentially) inferable facts can be classified into two categories: On the one hand, there are "useful" (*positive*) facts, which have to be inferred in order to follow the path given by $\mathcal{P}$. On the other hand, "useless" (*negative*) facts represent facts that lead away from this path, entailing redundant search effort. (We emphasize that the terms "positive" and "negative" must be seen in the context of finding the particular proof $\mathcal{P}$.) Consequently, an obvious method for estimating an ω consists in measuring its ability to distinguish positive facts from negative ones, i.e., its ability to cut off misleading paths. Here, we have to face another problem: While the set P of positive facts is finite (and usually rather small), the set of negative facts is in general infinite. Hence we have to confine ourselves to a finite subset N of the set of (all) negative facts. P and N can be extracted from the proof run that yielded $\mathcal{P}$, for instance. But there is one crucial problem in this approach, namely using a *static* N. Even in case we succeed in adapting $\mathcal{W}$ "perfectly", i.e., it associates weights[3] with the members of P and N so that the highest weight of a fact in P is still below the weight of any fact in N, we have absolutely no guarantee that such a perfect "adaptation" will carry over to an improved search for a proof. As a matter of fact, it is even possible for the adapted heuristic ω to perform more poorly than the heuristic originally employed to find $\mathcal{P}$. The reason for such a behavior is the high probability that ω may infer negative facts different from those in N, which might actually complicate the search for $\mathcal{P}$ even more. We therefore have to step back from the idea of using a *static* N.

The problems just outlined can—at least to a large extent—be compensated for by periodically updating N, i.e., by maintaining a *dynamic* N. The adaptive procedure (learning) then proceeds as follows: Starting with $N = \emptyset$, the automated prover is run for a given time T (usually about $5sec$) before calling the GA for the first time, and then each time the GA has executed n_c cycles. Each run through this outer loop (the inner loop corresponds to a cycle of the GA) will be referred to as an *iteration*. For these *update runs* the prover uses the ω currently rated best. If the prover actually succeeds in finding a proof, ω will be among the output of the adaptive procedure. Facts generated during time T which are not in P are considered as negative facts and are added to N. It must be emphasized that newly occurring (negative) facts are *added* to N and the "old" ones are *not* discarded. Otherwise, if replacing N each time, chances are that we might run into some form of "oscillation" where the same N appear cyclically, which may have severely disadvantageous effects on the GA's search for an optimum.

We have now explained the central role the GA plays regarding our learning task. We have also brought out that the central problematic of this GA application is the design of a fitness function. Through the discussion above it has become clear that—for efficiency and practicability reasons—we have to content ourselves with a fitness function that simulates rather than actually performs proof runs. This simulation is based on the sets P and N which depend on the source proof $\mathcal{P}$. We already pointed out that N has to be updated periodically. This measure is a remnant of the "ideal"

[3]It is assumed that a heuristic associates a weight (a natural number) with all potentially inferable facts and that the fact with the smallest weight is actually inferred.

fitness function which is indispensable to be able to achieve a "realistic" simulation. The following subsection describes the technical details of the fitness function.

3.2 Designing a fitness function

The fitness function of the GA works with the two sets $P = \{ \langle u_j, v_j \rangle \mid 1 \le j \le m \}$ and N, the latter being periodically updated after n_c cycles of the GA. In order to estimate the quality of an ω, i.e., its ability to prefer elements of P to elements of N, the following sets $\mathcal{I}_j^\omega(N) \subseteq N$ associated with each $\langle u_j, v_j \rangle \in P$ are pivotal. The elements of each $\mathcal{I}_j^\omega(N)$ are the *currently known* negative critical pairs which *may* be preferred to the respective $\langle u_j, v_j \rangle \in P$ during a search for $\mathcal{P}$ using ω.

$$\mathcal{I}_j^\omega(N) = \{ \langle u, v \rangle \in N \mid \omega(\langle u, v \rangle) \le \omega(\langle u_j, v_j \rangle) \}, \qquad 1 \le j \le m$$

A "perfect" adaptation entails $\mathcal{I}_j^\omega(N) = \emptyset$ for all $1 \le j \le m$. Since this will hardly ever be the case we have to find a more subtle way of rating a given ω. Basically, the number of elements in each $\mathcal{I}_j^\omega(N)$ constitute a lead, indicating the (potential) number of "obstacles" on the way to $\langle u_j, v_j \rangle$. Therefore our aim consists in minimizing $|\mathcal{I}_j^\omega(N)|$. But matters are not that simple that merely adding up the $|\mathcal{I}_j^\omega(N)|$ will do. The key idea is to estimate the (detrimental) influence of each $\langle u, v \rangle \in \mathcal{I}_j^\omega(N)$ for all $1 \le j \le m$. Essential for this estimation are the notions *relevance* and *irrelevance*. A negative critical pair $\langle u, v \rangle$ is *relevant (irrelevant) with respect to ω and $\langle u_j, v_j \rangle \in P$* if $\omega(\langle u, v \rangle) \le \omega(\langle u_j, v_j \rangle)$ and it is (is not) selected before $\langle u_j, v_j \rangle$ during a proof run using ω. (Recall that the members of N stem from "occasional" update runs with just one, namely the currently best ω'. They can therefore in no way be representative w.r.t. any ω.) It must be emphasized that relevance or irrelevance are not global properties of a negative critical pair $\langle u, v \rangle$, but must be seen with respect to a certain ω and $\langle u_j, v_j \rangle \in P$. A negative critical pair $\langle u, v \rangle$ may be relevant w.r.t. ω and $\langle u_j, v_j \rangle$, but may be irrelevant w.r.t. ω' and $\langle u_k, v_k \rangle$, or vice versa (where $\omega \not\equiv \omega'$ *or* $j \ne k$). It is clear that each irrelevant $\langle u, v \rangle \in \mathcal{I}_j^\omega(N)$ will make ω look worse than it actually is, unless it is identified as irrelevant. On the other hand, if a negative critical pair $\langle u, v \rangle$, which is relevant w.r.t. a certain ω and $\langle u_j, v_j \rangle \in P$, is not in N and consequently not in any $\mathcal{I}_j^\omega(N)$, then this ω might look better than it actually is. Relevance or irrelevance can only be decided if we really start a proof run using the respective ω. But this is exactly what we wanted to avoid. Therefore we have to use more efficient, but of course less accurate criteria to estimate relevance and irrelevance.

We can here merely sketch these criteria that are worked into the fitness function ϑ. (For a detailed discussion and the gradual development of ϑ see [Fu95].)

First of all, statistics concerning the frequency of occurrences of a $\langle u, v \rangle \in N$ during update runs give some clues regarding the relevance of $\langle u, v \rangle$. The total number of occurrences $\mu(\langle u, v \rangle) \le i_c$, where i_c is the number of the current iteration, as well as the number $i_{mr}(\langle u, v \rangle) \le i_c$ of the iteration $\langle u, v \rangle$ most recently occurred in are made use of in this context. Basically, the more occurrences and the more recently they took place, the more relevant $\langle u, v \rangle$ is considered to be. (See "occurrence component" below.)

A $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$ may have a weight $\omega(\langle u,v\rangle)$ smaller than or equal to $\omega(\langle u_j,v_j\rangle)$. Since $\langle u,v\rangle$ with $\omega(\langle u,v\rangle) = \omega(\langle u_j,v_j\rangle)$ is not necessarily an obstacle during the search for $\langle u_j,v_j\rangle$ even if it is in the set CP (FIFO policy), we distinguish these two cases. Hence, $\langle u,v\rangle$ with $\omega(\langle u,v\rangle) < \omega(\langle u_j,v_j\rangle)$ is considered more relevant than a $\langle u',v'\rangle$ with a weight equal to $\omega(\langle u_j,v_j\rangle)$. (See φ_j^ω below.)

Furthermore, the depth[4] $\delta(\langle u_j,v_j\rangle)$ of a $\langle u_j,v_j\rangle \in P$ in the derivation graph compared to the depth $\delta(\langle u,v\rangle)$ of a $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$ also gives some hints as to the relevance of $\langle u,v\rangle$: The deeper $\langle u,v\rangle$ compared to $\langle u_j,v_j\rangle$, the less relevant it is considered. (See "depth component" below.)

The criteria just presented are realized by the following formulas (in one of many possible ways), where $1 \le j \le m$, $\mathcal{D} \in \mathbb{N}$ and $\langle u,v\rangle \in \mathcal{I}_j^\omega(N)$.

$$\begin{aligned}
\varphi_j^\omega(\langle u,v\rangle) &= \begin{cases} \left\lfloor \frac{\mu(\langle u,v\rangle)+1}{2} \right\rfloor, & \text{if } \omega(\langle u,v\rangle) = \omega(\langle u_j,v_j\rangle) \\ \mu(\langle u,v\rangle), & \text{otherwise} \end{cases} \\
\psi_j^\omega(\langle u,v\rangle) &= \beta\Big(\underbrace{(\varphi_j^\omega(\langle u,v\rangle) - (i_c - i_{mr}(\langle u,v\rangle)))}_{\text{occurrence component}} \cdot \underbrace{\beta\left(\delta(\langle u_j,v_j\rangle) - \delta(\langle u,v\rangle) + \mathcal{D}\right)}_{\text{depth component}} \Big)
\end{aligned}$$

where $\beta(x) = x$ if $x > 0$ and 1 otherwise. Hence $\zeta_j^\omega = \sum_{\langle u,v\rangle \in \mathcal{I}_j^\omega(N)} \psi_j^\omega(\langle u,v\rangle)$ is a measure for the "difficulties" we (probably) have to face when using ω and trying to reach $\langle u_j,v_j\rangle \in P$. If ζ_j^ω grows these difficulties (are thought to) augment.

So far we have only provided criteria dealing with relevance resp. irrelevance of members of N. The last criterion aims at handling relevance of (negative) critical pairs not (yet) in N. We utilized the following simple, obvious observation: Chances that there are relevant $\langle u,v\rangle$ (w.r.t. a certain ω and a $\langle u_j,v_j\rangle \in P$) which are not in N and hence not in a $\mathcal{I}_j^\omega(N)$ increase *over-proportionally* (see α below) with $|\mathcal{I}_j^\omega(N)|$. We therefore count the elements of $\mathcal{I}_j^\omega(N)$ omitting those judged to be irrelevant on account of the above criteria giving

$$\kappa_j^\omega = \sum_{\langle u,v\rangle \in \mathcal{I}_j^\omega(N)} \pi\left(\varphi_j^\omega(\langle u,v\rangle) - (i_c - i_{mr}(\langle u,v\rangle))\right), \quad 1 \le j \le m$$

where $\pi(x) = 1$ if $x > 0$ and 0 otherwise.

The fitness function ϑ is a lexicographic combination of the four measures ϑ_A, ϑ_B, ϑ_C and ϑ_D. (The last two measures ϑ_C and ϑ_D compute average weights.)

$$\begin{aligned}
\vartheta_A(\omega) &= \sum_{j=1}^m \alpha\left(\kappa_j^\omega, \zeta_j^\omega\right), \quad \alpha(x,y) = \left\lfloor \frac{x^2 \cdot y}{\mathcal{C}} \right\rfloor, \quad \mathcal{C} > 0 \\
\vartheta_B(\omega) &= \sum_{j=1}^m \zeta_j^\omega \\
\vartheta_C(\omega) &= \frac{\sum_{j=1}^m \omega(\langle u_j,v_j\rangle)}{m} \qquad (m > 0)
\end{aligned}$$

[4]The depth of an axiom is 0. The depth of an inferred critical pair is the maximum of the depths of its ancestors plus 1. See also [Fu95].

$$\vartheta_D(\omega) \quad = \quad -\frac{\sum_{\langle u,v\rangle\in N}\omega(\langle u,v\rangle)}{|N|} \qquad (N \neq \emptyset)$$

Given two adapted heuristics ω and ω', ω is estimated to be better than ω' if $\vartheta(\omega) <_{lex} \vartheta(\omega')$, where $<_{lex}$ is the lexicographic comparison from left to right using the usual ordering $<$. This completes the description of the fitness function.

The subsequent section introduces our second method.

4 Extending a proof path

The second method stems from a more straight forward approach to utilizing past proof experience which has already been suggested in section 2: The first idea that comes to mind when thinking about reusing a proof $\mathcal{P}$ found in the past (the source proof) to prove a given theorem is to try to infer facts that are in some way similar to those needed for the source proof ("*derivational analogy*", e.g. [Ca86]). Unlike the method proposed in section 3, which diminishes flexibility for the sake of the benefits of increased specialization, we have to deal here with a method that is specialized and hence less flexible by design. Therefore, we have to walk on the opposite way, sacrificing a piece of specialization to obtain the indispensable degree of flexibility. This can be achieved by combining in an appropriate way the inference of similar facts with one of the general (flexible) heuristics the automated proving system at hand disposes of.

Basically, the combination consists in measuring the "distance" between a given fact and facts lying on the path prescribed by $\mathcal{P}$, which is overlapped by a general heuristic, the so-called *associated heuristic* ω. Usually, a potentially inferable fact λ is weighted by ω, i.e., ω associates with λ a natural number $\omega(\lambda) \in \mathbb{N}$, and the fact with the smallest weight is actually inferred. Here, we combine $\omega(\lambda)$ with a measure $d(\lambda)$ of the "distance" between λ and the facts whose (actual) inference allows to follow the path traced out by $\mathcal{P}$. Let $\mathcal{L}_\mathcal{P}$ be a list of those latter facts. Furthermore, a fact λ is either an axiom or has been inferred with the help of $n \geq 1$ facts $\lambda_1, \ldots, \lambda_n$ already known. In the latter case, we refer to $\lambda_1, \ldots, \lambda_n$ as the *ancestors* of the *descendant* λ.

We define the distance $d(\lambda)$ between a fact λ and the facts in $\mathcal{L}_\mathcal{P}$ as

$$d(\lambda) = \begin{cases} 0, & \lambda \text{ is an axiom (no ancestors)} \\ \psi(\gamma(\lambda), \mathcal{D}(\lambda)), & \lambda \text{ has } n \geq 1 \text{ ancestors} \end{cases}$$

γ computes the average *distance* between the ancestors $\lambda_1, \ldots, \lambda_n$ of λ and the facts in $\mathcal{L}_\mathcal{P}$, while $\mathcal{D}$ computes the (minimal) *difference* between λ itself and the facts in $\mathcal{L}_\mathcal{P}$. ψ combines these two values. We set

$$\gamma(\lambda) = \left\lfloor \frac{1}{n}\sum_{i=1}^{n} d(\lambda_i) \right\rfloor \quad , \qquad \psi(x,y) = \begin{cases} \left\lfloor \frac{x+y}{2} \right\rfloor, & y \neq 0 \\ 0, & y = 0 \end{cases} ,$$

$$\mathcal{D}(\lambda) = \min\left(\{ \mathit{diff}(\lambda, \lambda') \mid \lambda' \in \mathcal{L}_\mathcal{P}\}\right) .$$

diff is a (syntactical) difference measure which is inverse to similarity. We employ syntactical identity (modulo renaming variables) and subsumption as the most reliable and accurate meters of similarity:

$\lambda \approx \lambda'$ *iff* $\exists \rho : \rho(\lambda) \equiv \lambda'$, where ρ performs the renaming of variables

$\lambda \triangleleft \lambda'$ *iff* λ subsumes λ', which basically means testing 'λ *implies* λ'' (λ, λ' being closed formulas, i.e., not containing free variables) and this test can be decided on a syntactical level.

diff is now defined by

$$\mathit{diff}(\lambda, \lambda') = \begin{cases} 0, & \lambda \approx \lambda' \text{ or } \lambda \triangleleft \lambda' \\ 100, & \text{otherwise} \end{cases}$$

We restrict *diff* to $\mathbb{N}_{100} = \{0, \ldots, 100\}$, so that also all the other functions will return values from $\mathbb{N}_{100}$, which is a standardization that makes computations more transparent. So, both for d and *diff*, 0 stands for a "perfect agreement" with respect to $\mathcal{P}$, whereas 100 stands for a "total disagreement". Note that the distance 0 is assigned to axioms without checking for any similarity, which is justified by the observation that axioms have a high probability to be necessary for finding a proof (unless the set of axioms is deliberately or carelessly "overloaded"). Note also that the design of ψ causes the distance $d(\lambda)$ to increase depending on the average distance of its ancestors and its own difference, unless the latter is 0, which corresponds to λ being identical to or subsuming a fact in $\mathcal{L}_{\mathcal{P}}$.

The final weight $\varpi(\lambda)$ that this method associates with λ complies with the general convention, namely "*the smaller the weight of a fact, the more useful it is considered to be*".

$$\varpi(\lambda) = (d(\lambda) + p) \cdot \omega(\lambda) \quad , \quad p \in \mathbb{N}$$

The parameter p controls the effect of $d(\lambda)$, which will be dominant if $p = 0$. In that case, if also $d(\lambda) = 0$, then $\varpi(\lambda)$ will be 0, too, regardless of the weight $\omega(\lambda)$ the associated heuristic ω contributes. As p grows, $\omega(\lambda)$ increasingly influences the final weight, thus mitigating the inflexibility of the underlying method. For very large p, the influence of $d(\lambda)$ becomes negligible, and the whole method basically degenerates into the associated heuristic. Thus, ω attempts to effect an extension of $\mathcal{P}$ which d tries to confine, the whole interaction being controlled by p.

Before summarizing our experimental results in section 6, the subsequent section concisely describes the automated proving system used for these experiments. We have chosen an automated proving system for (purely) equational logic based on the Knuth–Bendix completion procedure (*UKB-procedure*). Our choice has mainly been favored by the existence of knowhow and implementations. We believe that our methods can be utilized by almost any (automated) prover that employs parameterized heuristics to control its inference machine and explicitly infers facts.

5 Equational theorem proving with the UKB-procedure

The unfailing Knuth-Bendix completion procedure ([KB70], [HR87], [BDP89]) is a procedure for purely equational reasoning. We shall now describe its foundations.

Let $\mathcal{F}$ be a finite set of function symbols (operators) and $\mathcal{V}$ an enumerable set of variables. $\tau : \mathcal{F} \rightarrow \mathbb{N}$ determines the arity of any $f \in \mathcal{F}$. The set $Term(\mathcal{F}, \mathcal{V})$ of terms over $\mathcal{F}$ and $\mathcal{V}$ is recursively defined by $\mathcal{V} \subseteq Term(\mathcal{F}, \mathcal{V})$, and, given $t_1, \ldots, t_n \in Term(\mathcal{F}, \mathcal{V})$, $f \in \mathcal{F}$, $\tau(f) = n$, then $f(t_1, \ldots, t_n) \in Term(\mathcal{F}, \mathcal{V})$. An equation is a pair of terms s and t, written as $s = t$. The UKB-procedure can be applied to investigate the following problem: Given a set $Ax = \{s_1 = t_1, \ldots, s_n = t_n\}$ of (implicitly) all-quantified equations (the *axioms*) and an also (implicitly) all-quantified equation $Th \equiv s = t$ (the *theorem* or *goal*), is Th a logical consequence of Ax? If this is the case, then the UKB-procedure can prove it by employing the usual operational semantics of equational deduction ("*replacing equals by equals*"), i.e., showing $Ax \vdash Th$.

The UKB-procedure was originally conceived for transforming the initial set of equations (Ax) into a confluent and terminating (i.e., convergent) set R of rewrite rules. Therefore, the UKB-procedure is forward-oriented by design. It proceeds by deriving new equations from those in the current set R (which is initially empty). Newly inferred equations are called *critical pairs* and are collected in the set CP. (Initially, $CP = Ax$.) During this inference process certain restrictions apply, most of which are based on the use of a *reduction ordering* $\succ$. $\succ$ is a partial ordering on terms which has among others the property to be well-founded. An outstanding advantage of the UKB-procedure is the simplification of the current sets CP and R. This eliminates much redundancy and also keeps the known facts as concise as possible. Rewriting steps (substituting terms with terms that are smaller w.r.t. $\succ$)—also called *reductions*—account for the majority of simplifications.

Equations in R are "active", i.e., they take part in the generation of critical pairs. The main loop of the UKB-procedure selects (and deletes) a critical pair *cp* from CP, which then becomes a member of R, performs all possible simplifications and adds all equations that can be inferred on account of *cp* to CP. A goal is proved if both its sides can be reduced to identical terms. Completion terminates if the set CP eventually becomes empty. Hence the crucial, indeterministic step of the UKB-procedure is the selection of the next critical pair to become active. A judicious choice on that score can speed up the proof (completion) process considerably, whereas poor choices can slow it down extremely and even make it (practically) impossible. Due to the general undecidability of $Ax \vdash Th$ only heuristics can be applied to resolve this indeterminism.

A heuristic for selecting the next critical pair usually associates a weight (a natural number) with each critical pair. Commonly (and that is what we assume here) the critical pair with the lowest weight is the one considered the most suitable and will be selected. (If there are several critical pairs with the same lowest weight then the one that has been in the set CP for the longest time is picked (FIFO).) The heuristics for choosing the next critical pair are based on a *weighting function* $\phi : \mathcal{F} \cup \mathcal{V} \rightarrow \mathbb{N}$, where $\phi(x) = w_v \in \mathbb{N}$ for all $x \in \mathcal{V}$ and $\phi(f) = w_f$ for all $f \in \mathcal{F}$. ϕ can be extended to $Term(\mathcal{F}, \mathcal{V})$ by defining

$$\phi(t) = \begin{cases} w_v, & \text{if } t \equiv x \in \mathcal{V} \\ w_f + \sum_{i=1}^{n} \phi(t_i), & \text{if } t \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F},\ n = \tau(f) \end{cases}$$

The heuristics *add* and *max* associate with a critical pair $\langle u, v\rangle$ the weights $\phi(u)+\phi(v)$ and $\max(\{\phi(u), \phi(v)\})$, respectively. A further heuristic, *occnest*, assigns

$$occnest_{s \neq t}(\langle u, v\rangle) = (\phi(u) + \phi(v)) \cdot \prod_{f \in \mathcal{F}} m_f$$

to a critical pair $\langle u, v\rangle$. *occnest* is goal oriented and therefore labeled with the current goal which is negated and skolemized in order to handle variables properly. The factors m_f express some structural difference between the critical pair $\langle u, v\rangle$ and $s \neq t$ for $f \in \mathcal{F}$ (see [DF94] or [Fu95] for details). The parameters of these heuristics are the values w_v and w_f for all $f \in \mathcal{F}$. The *default versions* of these three heuristics (as used by our system) can be obtained by choosing $w_v = 1$ and $w_f = 2$ for all $f \in \mathcal{F}$.

6 Experimental results

In this section an excerpt of the results of our experimentation will be presented. The entries in the bodies of all subsequent tables display run-times (in seconds), obtained on a SPARCstation 1. The equational prover used for these experiments, namely the DISCOUNT system ([ADF95]), is written in C. The tools for computing the set P needed by the learning method as well as the set $\mathcal{L}_{\mathcal{P}}$ employed by 'path extension' are based on existing software for proof extraction and analysis (see [DS94a], [DS94b]). Each of the following subsections 6.1 and 6.2 deals with a different set of (proof or completion) problems. In order to be able to estimate the achievements of our two methods presented in sections 3 and 4, we shall also list the results produced by the default versions of the heuristics *add*, *max* and *occnest* (cf. section 5) to provide a point of reference. Whenever the method 'path extension' (cf. section 4) was employed we set $p = 30$ and the associated heuristic $\omega \equiv add$ (default version). It will be explained by way of example how to interpret the tables to come.

6.1 The first set of problems: propositional logic

These problems go back to [Ta56]. The set of axioms Ax^{PL} is given by

$$\begin{array}{rclrclrcl} c(t,x) & = & x & c(n(n(x)),x) & = & t & c(c(x,c(y,z)),c(y,c(x,z))) & = & t \\ c(x,c(y,x)) & = & t & c(x,n(n(x))) & = & t & c(c(x,y),c(n(y),n(x))) & = & t \end{array}$$
$$c(c(x,c(y,z)),c(c(x,y),c(x,z))) = t$$

and is an equational axiomatization of the propositional logic. ('c' corresponds to 'implication', 'n' to 'not' and 't' to 'true'.) Consequently, the theorems are tautologies of the propositional logic.

	$\mathcal{A}_1^{PL}$	$\mathcal{A}_2^{PL}$	$\mathcal{A}_3^{PL}$	$\mathcal{A}_4^{PL}$	$\mathcal{A}_5^{PL}$	$\mathcal{A}_6^{PL}$	$\mathcal{A}_7^{PL}$	$\mathcal{A}_8^{PL}$
add	276s	276s	340s	330s	318s	333s	321s	318s
$occnest$	14s	14s	67s	65s	69s	9.6s	9.4s	29s
$\omega_1^{occnest}$	7.74s	7.45s	17s	17s	17s	8.2s	8.3s	31s
$\mathcal{P}_1$	0.74s	0.70s	1.34s	0.74s	1.28s	1.34s	1.29s	0.66s
$\mathcal{P}_7$	0.30s	0.32s	0.54s	0.31s	0.34s	0.52s	0.34s	0.93s
$\mathcal{P}_8$	0.87s	0.82s	0.99s	0.86s	0.93s	0.99s	0.93s	0.28s

Table 6.1.1

$$
\begin{array}{lcl}
Th_1^{PL} & \equiv & c(x, c(n(x), y)) = t \\
Th_2^{PL} & \equiv & c(n(x), c(x, y)) = t \\
Th_3^{PL} & \equiv & c(x, c(n(y), c(n(x), z))) = t \\
Th_4^{PL} & \equiv & c(x, c(n(x), c(n(y), z))) = t
\end{array}
\qquad
\begin{array}{lcl}
Th_5^{PL} & \equiv & c(n(x), c(n(y), c(x, z))) = t \\
Th_6^{PL} & \equiv & c(x, c(y, c(n(x), z))) = t \\
Th_7^{PL} & \equiv & c(n(x), c(y, c(x, z))) = t \\
Th_8^{PL} & \equiv & c(n(c(n(x), y)), n(x)) = t
\end{array}
$$

For *any* problem $\mathcal{A} = (Ax^{PL}, Th)$ the heuristics add and max agree completely. Therefore, the first two rows of table 6.1.1 merely list the results obtained with add and $occnest$. For this set of problems the method 'path extension' (section 4) is clearly superior to the 'learning' method (section 3). For the latter, table 6.1.1 shows one result only, namely $\omega_1^{occnest}$ in the third row, which was generated by adapting $occnest$ to the proof of $\mathcal{A}_1^{PL} = (Ax^{PL}, Th_1)$. $\omega_1^{occnest}$ is among the more successful instances obtained by learning. The remaining rows exhibit the achievements of 'path extension'. Consider, for instance, row five which is headed by $\mathcal{P}_7$. The label $\mathcal{P}_7$ means that the path given by the (known) proof of $\mathcal{A}_7^{PL}$ was the basis when proving, e.g., $\mathcal{A}_4^{PL}$ in $0.31sec$. The salient speed-ups are also present when using the proofs of $\mathcal{A}_2^{PL}, \ldots, \mathcal{A}_6^{PL}$ not displayed by table 6.1.1. The main reason why 'learning' is here inferior to 'path extension' resides in the fact that the (known) proofs of the above theorems all involve critical pairs that have a rather large number of function symbols. Since the heuristics used here heavily depend on the number of function symbols, it is impossible to find parameter configurations that allow to reduce sufficiently the number of (negative) critical pairs receiving a lesser weight due to a considerably lesser number of function symbols.

6.2 The second set of problems: completion tasks

The problems dealt with in this subsection are completion tasks (cp. [Ch93], [Zh92]). The initial set of equations for such a task $\mathcal{A}_n$ is

$$
\begin{array}{ll}
f(e_j, x) = x \quad , & 1 \le j \le n \\
f(x, i_j(x)) = e_j \quad , & 1 \le j \le n \\
f(f(x, y), z) = f(x, f(y, z)) \ . &
\end{array}
$$

	$\mathcal{A}_8$	$\mathcal{A}_{10}$	$\mathcal{A}_{20}$	$\mathcal{A}_{30}$	$\mathcal{A}_{40}$	$\mathcal{A}_{50}$	$\mathcal{A}_{60}$	$\mathcal{A}_{70}$	$\mathcal{A}_{80}$	$\mathcal{A}_{90}$	$\mathcal{A}_{100}$
add	1.6s	2.9s	27.0s	110s	340s	850s	∞	∞	∞	∞	∞
max	3.9s	8.0s	93.8s	1120s	4600s	∞	∞	∞	∞	∞	∞
Herky	4.2s	6.8s	25.0s	79.9s	179s	323s	623s	1031s	1610s	2238s	2964s
$\mathcal{P}_8$	0.20s	0.34s	2.46s	10.2s	28.6s	68.2s	139s	254s	429s	684s	1043s
$\mathcal{P}_{50}$	0.36s	0.47s	1.36s	2.80s	4.86s	7.27s	18.3s	46.9s	104s	202s	360s
ω_8^{max}	0.23s	0.28s	0.69s	1.37s	2.16s	3.18s	4.16s	5.34s	6.87s	8.56s	10.1s

Table 6.2.1

Our completion process yields $3n+6$ rules, requiring the generation of at least $4n+7$ rules, if the reduction ordering is a LPO with precedence $i_n \succ_p \cdots \succ_p i_1 \succ_p f \succ_p e_n \succ_p \cdots \succ_p e_1$. We consider here the eleven instances obtained by choosing $n \in \{8, 10, 20, \ldots, 100\}$. Note that it still makes sense to talk about proofs in this context if we view the completion process as proving the rules of the resulting convergent system. The first two rows of table 6.2.1 list the results of the default heuristics *add* and *max*. We omitted *occnest* because it does not make sense to use a goal oriented heuristic for completion. The entry '∞' indicates that no convergent system could be produced because of memory shortage. The third row shows the results reported in [Zh92] produced by Herky, which is one of the most powerful equational provers currently available (see also [Zh93]). Note that Herky is implemented in LISP, but can nevertheless defy serious C implementations (e.g., OTTER) w.r.t. many equational problems (including those considered here). The fourth row displays the results obtained when employing the method 'path extension' (section 4) with $\mathcal{L}_\mathcal{P}$ set to the list of all critical pairs necessary to generate a convergent set of rules starting with the simplest of these tasks, namely $\mathcal{A}_8$. (This row is therefore headed by $\mathcal{P}_8$). It is obvious that we can this way already achieve considerable improvements that culminate in a successful completion of $\mathcal{A}_{60}, \ldots, \mathcal{A}_{100}$ which is not possible when using default heuristics. Nonetheless, it remains an imperfection that any critical pair occurring during the completion process of $\mathcal{A}_n$ $(n > 8)$ that contains a function symbol not occurring in $\mathcal{A}_8$ $(i_9, \ldots, i_n$ or $e_9, \ldots, e_n)$ is considered as "totally different" (i.e., the value 100 is assigned to it by $\mathcal{D}$). This is certainly not satisfactory from an analogical perspective. The results produced when this method rests on the completion of $\mathcal{A}_{50}$ (denoted by $\mathcal{P}_{50}$ in the fifth row) illustrate this disadvantage by showing that the additional information concerning the function symbols $i_9, \ldots, i_{50}$ and $e_9, \ldots, e_{50}$ clearly pays off with significantly lesser run-times.

When employing our 'learning' method (section 3), adapting *max* to the completion of $\mathcal{A}_8$, we obtain ω_8^{max} displayed in the last row of table 6.2.1. ω_8^{max} impressively outperforms everything including Herky. The problematic concerning additional function symbols is resolved in compliance with the principle of this method which consists in assigning "appropriate" values to the parameters of *max*. Therefore, when applying ω_8^{max} to the completion of $\mathcal{A}_n$ $(n > 8)$, the average weight of the weights $w_{i_1}, \ldots, w_{i_8}$ and the average of $w_{e_1}, \ldots, w_{e_8}$ is assigned to $w_{i_9}, \ldots, w_{i_n}$ and $w_{e_9}, \ldots, w_{e_n}$ respectively.

This way of proceeding is of course merely a heuristic itself. But it is justifiable and easy to automate. In order to achieve a similar effect with 'path extension' a thorough analysis of the completion process together with far-reaching changes of *diff* would be necessary, all of which is not remotely as easy to automate. Furthermore, although $\mathcal{P}_{50}$ works "optimally" for completing $\mathcal{A}_n$ with $n \leq 50$, generating the minimal number of rules $(4n+7)$, it is inferior to $\omega_{\mathrm{g}}^{max}$ regarding run-times. $\omega_{\mathrm{g}}^{max}$, which is very close to the optimum, generating $4n+13$ rules for *all* $\mathcal{A}_n$, causes no overhead during its application, whereas $\mathcal{P}_{50}$ has to submit each critical pair to $|\mathcal{L}_{\mathcal{P}_{50}}|$ rather time consuming tests necessary to compute $\mathcal{D}$. (In case of $\mathcal{P}_{50}$, $|\mathcal{L}_{\mathcal{P}_{50}}| = 4 \cdot 50 + 7 = 207$.) The time spent by the adaptive procedure to produce $\omega_{\mathrm{g}}^{max}$ was ca. $3min$. But we think that this time is not a real issue, because adaptation can be done once and for all during the "spare time" of the proving system.

7 Related work

Poor performance of automated proving systems when it comes to proving difficult theorems, and the resulting lack of competitiveness compared to mathematicians are the main reasons to improve the heuristics of the respective provers. Although most designers of automated provers have recognized that the best and most natural way to do this is by learning from previous successfully solved tasks (e.g., [BCP88], [Bu88] and more recently [KW94]) the number of reports on *automated* approaches to learning in this environment *substantiated by experimental results* is rather low. To our knowledge, there has so far no work been done aiming at integrating an automated learning component into a prover for purely equational logic. We are aware of two research papers dealing with such a component for provers for first order logic ([SF71] and [SE90]). Common to both approaches is the representation of knowledge as clauses (CNF), and both use so-called *features* of clauses (see also [Su90]) as the basis of learning.

Features reflect certain properties of clauses, e.g., the total number of literals, the number of positive and negative literals etc. In [SF71], *feature vectors* are extracted from the data provided by a successful proof, each feature vector belonging to one clause derived during search. "Profit values" are associated with each feature vector, each expressing in some respect the usefulness of the related clause. Learning consists in finding functions approximating the 'feature vector vs. profit value'–relation. These functions are linear polynoms (in the features) whose coefficients are determined via multiple regression analysis (see [SF71] for details).

[SE90] also use feature vectors which are—properly encoded—presented to a neural network. The desired input–output behavior of the net is extracted from successful proofs (The only output unit is supposed to produce 1 for "useful" clauses, 0 for "useless" ones).

Both approaches will not work when applied to pure equational deduction unless additional features are introduced. Most of the current features are not distinctive when the clauses are limited to unit clauses with the equality predicate as the only predicate. Although we demonstrated the power of both our methods with an equational theorem prover, it should have become clear that their applicability is not restricted to this kind of prover at all. As a matter of fact, any theorem prover employing parameterized heuristics and explicitly inferring facts can profit from these methods.

In the general context of proof reuse also the work reported in [KW94] ought to be mentioned. The approach taken there combines ideas from explanation based learning and analogical reasoning as well as abstraction techniques. It consequently differs substantially from our approach, since proof reuse is accomplished by analyzing and generalizing proofs found in the past, which are then—properly instantiated—utilized for finding similar proofs. In [KW94], the notion 'similarity' is clearly defined, but proofs have to be very similar in this sense in order for this approach to be successful or applicable.

8 Summary

We have presented two methods that enable an automated proving system to make use of proof experience gained in the past. The indispensable compromise regarding generality (flexibility) and specialization (inflexibility, rigidity) of these methods is obtained in distinct ways. The first method starts with a general and flexible heuristic, achieving the desired degree of specialization by learning the parameters of this heuristic based on experience from the past. The second method, which attempts to re-enact a given proof and hence is very specialized by design, attains flexibility by integrating a general, flexible heuristic into its framework.

The experiments conducted so far in the area of equational reasoning, which are reported here only in part, have illustrated in a promising way the potential of both methods. We emphasize that we not only achieved substantial speed-ups, but also made it possible to handle problems that seemed out of reach before.

Finally, this report deals with the problematic *how* to make use of a known proof. Future work has to concentrate on an inherent difficulty of proof reuse not addressed in this report, namely to determine *when* to apply experiences gained in the past. A judicious decision on that score is crucial in order to avoid a major pitfall of proof reuse which consists in sometimes causing a change for the worse compared to proving from scratch (cp. [KN93]).

References

[ADF95] **Avenhaus, J.; Denzinger, J.; Fuchs, M.:** *DISCOUNT: A system for distributed equational deduction*, to appear in Proc. 6^{th} RTA, Kaiserslautern, FRG, 1995

[BCP88] **Brock, B.; Cooper, S.; Pierce, W.:** *Analogical reasoning and proof discovery*, Proc. CADE 9, Argonne, IL, USA, 1988, LNCS 310, pp. 454–468

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

[Bu88] **Bundy, A.:** *The use of explicit plans to guide inductive proofs*, Proc. CADE 9, Argonne, IL, USA, 1988, LNCS 310, pp. 111–120

[Bu89] **Burstein, M.H.:** *Analogy vs. CBR: The purpose of mapping*, in: K.J. Hammond (ed.), Proceedings: Second case-based reasoning workshop (DARPA), Morgan Kaufmann, San Mateo, CA, USA, 1989, pp. 133–136

[Ca86] **Carbonell, J.:** *Derivational analogy: a theory of reconstructive problem solving and expertise acquisition*, in: R.S. Michalski et al. (eds.), Machine Intelligence; an AI approach, Vol. 2, 1986, pp. 371–392

[Ch93] **Christian, J.:** *Flatterms, discrimination nets, and fast term rewriting*, JAR 10, 1993, pp. 95–113

[Da88] **Davis, L. (ed):** *Genetic algorithms and simulated annealing*, Research notes in artificial intelligence, 1988

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

[DS94a] **Denzinger, J.; Schulz, S.:** *Analysis and Representation of Equational Proofs Generated by a Distributed Completion Based Proof System*, SEKI-Report SR-94-05, University of Kaiserslautern, 1994

[DS94b] **Denzinger, J.; Schulz, S.:** *Recording, Analyzing and Presenting Distributed Deduction Processes*, Proc. PASCO '94, Linz, Austr., 1994

[Fu95] **Fuchs, M.:** *Learning proof heuristics by adapting parameters*, to appear as SEKI-Report SR-95-XX, University of Kaiserslautern, 1995

[Ho75] **Holland, J.H.:** *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, Ann Arbor: Univ. of Michigan Press, 1975

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

[KB70] **Knuth, D.E.; Bendix, P.B.:** *Simple Word Problems in Universal Algebra*, Computational Algebra, J. Leech, Pergamon Press, 1970, pp. 263–297

[KN93] **Koehler, J.; Nebel, B.:** *Plan modification versus plan generation*, Proc. IJCAI '93, Chambery, FRA, 1993, pp. 1436–1444

[Ko92] **Kolodner, J.L.:** *An introduction to case-based reasoning*, Artificial Intelligence Review 6, 1992, pp. 3–34

[KW94] **Kolbe, T.; Walther, C.:** *Reusing proofs*, Proc. 11^{th} ECAI '94, Amsterdam, HOL, 1994, pp. 80–84

[Mc94] **McCune, W.W.:** OTTER *3.0 reference manual and guide*, Techn. report ANL-94/6, Argonne Natl. Laboratory, 1994

[Ra91] **Rawlins, G. (ed):** *Foundations of genetic algorithms*, Morgan Kaufmann, 1991

[SE90] **Suttner, C.; Ertel, W.:** *Automatic acquisition of search guiding heuristics*, Proc. CADE 10, Kaiserslautern, FRG, 1990, LNAI 449, pp. 470–484

[SF71] **Slagle, J.R.; Farrell, C.D.:** *Experiments in automatic learning for a multipurpose heuristic program*, Communications of the ACM, Vol. 14, Nr. 2, 1971, pp. 91–99

[Su90] **Suttner, C.:** *Representing heuristic-relevant information for an automated theorem prover*, Proc. 6^{th} IMYCS: aspects and prospects of theoretical computer science, LNAI 464, 1990

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

[Zh92] **Zhang, H.:** Herky*: High performance rewriting in RRL*, Proc. CADE 11, Saratoga Springs, NY, USA, 1992, LNAI 607, pp. 696–700

[Zh93] **Zhang, H.:** *Automated proofs of equality problems in Overbeek's competition*, JAR 11, 1993, pp. 333–351