Experiments in the Automatic Selection of Problem-solving Strategies*

Matthias Fuchs
Centre for Learning Systems and Applications (LSA)
Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
Germany
E-mail: fuchs@informatik.uni-kl.de

September 26, 1996

Abstract

We present an approach to automating the selection of search-guiding heuristics that control the search conducted by a problem solver. The approach centers on representing problems with feature vectors that are vectors of numerical values. Thus, similarity between problems can be determined by using a distance measure on feature vectors. Given a database of problems, each problem being associated with the heuristic that was used to solve it, heuristics to be employed to solve a novel problem are suggested in correspondence with the similarity between the novel problem and problems of the database.

Our approach is strongly connected with instance-based learning and nearest-neighbor classification and therefore possesses incremental learning capabilities. In experimental studies it has proven to be a viable tool for achieving the final and crucial missing piece of automation of problem solving—namely selecting an appropriate search-guiding heuristic—in a flexible way.

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

1 Introduction

Problem solving in artificial intelligence often amounts to search. A problem solver generally needs to employ some problem-solving strategy which in the common case that the problem solver conducts a search means that it needs a *search-guiding strategy* or *heuristic*. Most interesting and demanding problems require substantial search effort that must be kept within acceptable limits by using sophisticated search-guiding heuristics.

Unfortunately, there is no such thing as a "universal heuristic": Different problems (possibly even from the same small problem class) necessitate different heuristics in order to solve them in acceptable time. That is, a problem P to be solved must be associated with an appropriate heuristic that allows to solve P (in acceptable time). This crucial and difficult task is mostly taken on by the user himself. This missing piece of automation is—besides being unsatisfactory—putting off potential users since it requires expertise and experience, which becomes most apparent in connection with automated deduction. There, even experts of the field have difficulties and need some time to get familiar and to work effectively with an automated deduction system other than their own.

The well-known theorem prover OTTER alleviates this problem with its 'autonomous mode' (cp. [Mc94]) that attempts to pick an appropriate search-guiding heuristic after analyzing the current problem. But the "selection heuristic" is built-in and hence inflexible in the sense that it encodes the knowledge its designers had at some point in time. We, however, intend to automate choosing an appropriate heuristic by means of a database of previously solved problems. That is, we provide machine-learning capabilities that make it possible for an automated deduction system (respectively a problem solver in general) to profit from its problem-solving experience. Each problem P' in the database is associated with a heuristic H' that is the best (known) heuristic for solving P'. Given a problem P to be solved, a heuristic H to solve it is picked on the basis of the similarity between P and problems in the database.

To this end, each problem is represented by its *feature vector*. Each component of a feature vector is a *feature value*, i.e., a numerical value that represents a property (i.e., a feature) of the respective problem. Similarity between problems is defined in terms of a distance measure regarding their feature vectors. The heuristic to be applied to solve a problem P is determined with the help of the *nearest-neighbor rule* (NNR, [CH67]): The heuristic associated with the problem most similar (i.e., nearest) to P is selected.

Naturally, the choice of features is crucial for a sensible estimation of similarity. For our experiments in connection with automated deduction, however, a rather small number of obvious and easy to define features sufficed to obtain satisfactory results. At this point we would like to emphasize that our approach should be applicable to all kinds of problem solvers although we experimented solely in the area of automated deduction. But this is more a challenge than a restriction: Automated deduction not only poses some of the hardest (search) problems, but also has the property that often tiny variations of a problem description can cause significant changes in the solution, and therefore require different search-guiding heuristics, which makes choosing

an appropriate heuristic based on the similarity of problems even harder.

This report is organized as follows. Section 2 reviews the fundamentals and explains details of our NNR-based approach. Our experiments are documented in section 3. A discussion in section 4 concludes this report.

2 Details of the NNR-based Approach

Section 1 already sketched our approach in the context of automated deduction. In this section, we shall review and give details of our approach for problem solving based on search in general (which of course includes automated deduction).

2.1 Basics

The major difficulty from a user's point of view when working with a problem solver (based on search) is to select an appropriate search-guiding heuristic that allows him to solve his problem in acceptable time (where "acceptable" both depends on the user's demands and on the problem domain).

Usually, a problem solver PS has $m \geq 1$ search-guiding heuristics $H_1, \ldots, H_m$ at its disposal. Given a problem P to be solved, a $H \in \{H_1, \ldots, H_m\}$ must be chosen that guides the search conducted by PS. The quality of this choice is reflected by the time PS needs to solve P when employing H.

When dealing with search problems—in particular in connection with automated deduction—it is often the case that the discrepancy between an appropriate and an inappropriate heuristic is enormous: An appropriate heuristic may solve a problem within a couple of seconds, whereas an inappropriate heuristic may take several hours or even days. Since neither human nor automated selection of a heuristic can guarantee that *always* the (most) appropriate heuristic is selected, we adopt the following common procedure: Instead of choosing a single heuristic, a sequence S of heuristics is set up. The heuristics are then tried out in the order given by S. As a consequence, a maximal run time, i.e., a *time-out* T must be imposed on each heuristic in order to apply this principle in a purposeful way. Hence, the time-out is a (rather reasonable) means to anticipate failure to succeed in acceptable time. If a heuristic exceeds T, a new attempt is started with the next heuristic as given by S.

2.2 Assessing the Performance of a Sequence of Heuristics

Given the set $\{H_1, \ldots, H_m\}$ of available heuristics, a sequence S is represented by a permutation σ of $\{1, \ldots, m\}$, i.e., $S = S_\sigma \equiv H_{\sigma(1)}, \ldots, H_{\sigma(m)}$. When being confronted with $n \geq 1$ problems $P_1, \ldots, P_n$, a sequence S_{σ_i} must be set up for each problem P_i. Assuming that the time required by heuristic H_j to solve P_i is t_{ij}, where $t_{ij} = T$ if

H_j cannot solve P_i before time-out, and $0 < t_{ij} < T$ otherwise, the time S_{σ_i} needs to solve P_i is $\tau_i(\sigma_i)$, where

$$\tau_i(\sigma) = \sum_{j=1}^{q_i(\sigma)} t_{i\sigma(j)},$$

and $q_i(\sigma)$ indicates the ordinal of the first heuristic of S_σ that succeeds in solving P_i before time-out, i.e.,

$$q_i(\sigma) = \min(\{j \mid t_{i\sigma(j)} < T\} \cup \{m\}).$$

(Note that $\tau_i(\sigma) = m \cdot T$ if none of the heuristics $H_1, \ldots, H_m$ can solve P_i before time-out.) Thus,

$$\mathcal{T}_V = \sum_{i=1}^{n} \tau_i(\sigma_i)$$

is the *total time* required to solve $P_1, \ldots, P_n$. $\mathcal{T}_V$ is used to measure the performance of an approach to setting up sequences of heuristics, and it will be used to assess the performance of our NNR-based approach.

The index "V" of $\mathcal{T}_V$ signifies that the sequences involved are *variable* in the sense that they may differ from problem to problem, as opposed to the special case where a single *fixed* sequence S_{σ_F} is applied to all problems resulting in a total time

$$\mathcal{T}_F = \sum_{i=1}^{n} \tau_i(\sigma_F).$$

Besides such a fixed sequence, we shall also use the optimal total time $\mathcal{T}_{min}$ as a point of reference, where

$$\mathcal{T}_{min} = \sum_{i=1}^{n} \min(\{t_{ij} \mid 1 \leq j \leq m\}).$$

(Naturally, we have $\mathcal{T}_V, \mathcal{T}_F \geq \mathcal{T}_{min}$ regardless of σ_F and $\sigma_1, \ldots, \sigma_n$.)

2.3 Determining Variable and Fixed Sequences

Both the variable sequences and the fixed sequence are determined with the help of previous problem-solving experience given in form of problems that are each paired with a heuristic that is the best (known) to solve the respective problem. Hence, we have a database of pairs $(P_i, H_{b(i)})$, where $b(i) \in \{1, \ldots, m\}$ denotes the best heuristic (among $H_1, \ldots, H_m$) for solving P_i, i.e., $t_{ib(i)} = \min(\{t_{ij} \mid 1 \leq j \leq m\})$.

In order to evaluate the performance of our approach, a certain subset of given problems $P_1, \ldots, P_n$ will be considered to be previous experience. That is, problems in $\{P_i \mid i \in I\}$ for some $I \subseteq \{1, \ldots, n\}$ are regarded as *training problems* that constitute the database $DB_I = \{(P_i, H_{b(i)}) \mid i \in I\}$. Fixed sequence and variable sequences are determined with the help of DB_I. Their performance is measured in terms of the total times $\mathcal{T}_F$ and $\mathcal{T}_V$ required to solve all n problems. Since DB_I (respectively I) naturally has a crucial influence on performance, tests will be made with respect to various (randomly chosen) I, and we shall use best, worst, and average cases for assessment (cp. section 3).

Based on DB_I, the fixed sequence S_{σ_F} that is optimal for solving all training problems satisfies

$$\sum_{i \in I} \tau_i(\sigma_F) \leq \sum_{i \in I} \tau_i(\sigma)$$

for all permutations σ. For our experiments (section 3), S_{σ_F} will be employed as a competitor for the variable sequences in order to demonstrate that the problems tackled there are not so uniform that a fixed sequence can be a match for variable sequences that may vary depending on the problem to be solved.

Variable sequences are determined according to our NNR-based approach: Given a problem P_j to be solved, DB_I is searched for those P_i ($i \in I$) that are most similar to P_j. In order to make the assessment of similarity amenable to computation, a problem P_i is represented by a *feature vector* $\nu(P_i) = (v_1^i, \ldots, v_k^i)$. Each v_l^i is a *feature value* of a feature f_l, i.e., $v_l^i = f_l(P_i)$, $1 \leq l \leq k$. The features $f_1, \ldots, f_k$ each represent a property of a problem P_i with a numerical value, i.e., the feature value. Similarity can then be assessed by using a distance measure $\mathcal{D}$. The distance (similarity) between a problem P_j to be solved and a problem P_i of the database is then given by

$$\delta_j(i) = \mathcal{D}(\nu(P_j), \nu(P_i)), \quad i \in I.$$

The sequence S_{σ_j} for solving P_j should be set up so that the heuristics tried out first are the ones associated with the problems P_i ($i \in I$) that are most similar (nearest) to P_j. That is we compute $\delta_j(i)$ for all $i \in I$, and then sort these distances so that $\delta_j(i_1) \leq \cdots \leq \delta_j(i_r)$, where $\{i_1, \ldots, i_r\} = I$. From this we derive the sequence $\hat{S}_{\sigma_j} = H_{b(i_1)}, \ldots, H_{b(i_r)}$ from which we remove all multiple occurrences of the same heuristic except for the respective first occurrence. To the resulting sequence, heuristics $H \in \{H_1, \ldots, H_m\}$ not appearing in it are appended to it in arbitrary order, which finally produces S_{σ_j}.

The following section will show that this rather simple approach achieves remarkable results. Naturally, its performance is crucially affected by the choice of features and the distance measure. But again, rather simple and intuitive choices on that score were sufficient in our experiments, although the chosen problem domain, namely automated deduction, must be considered one of the most difficult domains in every respect.

3 Experiments

We conducted our experiments in equational reasoning, an important and difficult branch of automated deduction. Our problem solver hence is a system for equational reasoning. We employed the DISCOUNT system ([ADF95]) that is based on the unfailing Knuth–Bendix completion procedure ([BDP89]). The problems are taken from the public TPTP problem library version 1.2.0 ([SSY94]). More precisely, we selected a total of 264 problems from six problem classes `BOO`, `COL`, `GRP`, `LCL`, `RNG`, and `ROB`. (These abbreviations are used and explained by the TPTP library.)

For our experiments, DISCOUNT had $m = 5$ heuristics at its disposal.[1] Since the working method of these heuristics is here immaterial, they will henceforth be "anonymously" denoted by $H_1, \ldots, H_5$. (They are all based on the common weighting respectively counting of symbols.) Given the time-out $T = 600$ seconds, for each of the 264 problems there was at least one $H \in \{H_1, \ldots, H_5\}$ that allowed DISCOUNT to find a proof before time-out (on a SPARCstation 10). That is, we omitted problems that DISCOUNT could not prove before time-out no matter which one of $H_1, \ldots, H_5$ it employed as its search-guiding heuristic. This makes sense, because each such problem would simply increase the total times $\mathcal{T}_F$ and $\mathcal{T}_V$ by $m \cdot T = 3000$ seconds regardless of fixed or variable sequences used, and therefore does not help in assessing the performance of our approach.

Obviously, the time-out T affects $\mathcal{T}_V$ and $\mathcal{T}_F$, because every heuristic employed that does not succeed before time-out increases $\mathcal{T}_V$ respectively $\mathcal{T}_F$ by T seconds. Therefore, a large T will make differences between the performance of fixed and variable sequences much more noticeable than a small T. A large T, however, is in practice not desirable, because too much time is spent on runs that will finally time out in the hardly avoidable event that inappropriate heuristics occur first in a suggested (variable or fixed) sequence. Moreover, increasing T beyond a certain value augments the number of problems that can be solved before time-out only marginally, whereas of course too small a time-out diminishes that number dramatically. Thus, $T = 600$ seconds must be considered to be a compromise (regarding DISCOUNT) that attempts to take into account all of these aspects.

Problems of equational reasoning are specified like problems of automated deduction in general by giving a set Ax of axioms and the goal λ to be proved. In equational reasoning, all axioms are equations, i.e., $Ax = \{s_1 = t_1, \ldots, s_p = t_p\}$. Both sides s_i, t_i of an equation $s_i = t_i$ are first-order terms composed of variables and function symbols. A goal is a negated and Skolemized equation $s \neq t$. (Variables are implicitly all-quantified.)

Hence, each problem $P_i = (Ax_i, \lambda_i)$. We deployed the following $k = 10$ rather obvious and simple features $f_1, \ldots, f_{10}$ to obtain a feature vector $\nu(P_i)$ of P_i.

- f_1: $|Ax_i|$, i.e., the number of axioms;
- f_2: number of distinct function symbols occurring in axioms only;
- f_3: number of distinct constants occurring in axioms only;
- f_4: number of distinct unary function symbols occurring in axioms only;
- f_5: number of distinct binary function symbols occurring in axioms only;
- f_6: number of distinct ternary function symbols occurring in axioms only;
- f_7: number of distinct function symbols occurring in goal λ_i only;

[1] For these experiments DISCOUNT always used a LPO as reduction ordering with a precedence determined automatically according to some simple heuristic criteria.

Table 1: All problems are training problems ("recall precision")

Name	#	$\mathcal{T}_{min}$	$\mathcal{T}_V$	$\mathcal{T}_F$	1^{st} var.	1^{st} fixed
BOO	30	465 s	469 s	1494 s	100%	96.67%
COL	70	1270 s	5809 s	10936 s	95.71%	84.29%
GRP	108	1597 s	1707 s	13802 s	100%	85.19%
LCL	24	561 s	562 s	1775 s	100%	91.67%
RNG	22	165 s	165 s	2136 s	100%	86.36%
ROB	10	186 s	786 s	235 s	90%	100%
all	264	4244 s	9502 s	43913 s	98.48%	85.93%

- f_8: number of distinct variables occurring in goal λ_i;
- f_9: total number of symbols occurring in the smallest side of λ_i;
- f_{10}: total number of symbols occurring in the biggest side of λ_i;

Thus, all features produce a natural number. In order to assess the similarity of respectively to measure the distance between feature vectors we chose the distance measure $\mathcal{D}$ to be the Euclidean distance.

Table 1 shows $\mathcal{T}_{min}$, $\mathcal{T}_V$, and $\mathcal{T}_F$ for each problem class separately (BOO,...,ROB) and for all problems together ("all") when *all* respective problems are included in the database DB_I. That is, 100% of the respective problems are considered to be training problems. Hence, table 1 in a way displays "recall precision". The column labeled '#' lists the respective number of problems. Columns "1^{st} var." and "1^{st} fixed" show the percentage of problems solved by the first heuristic of the respective variable and fixed sequences. In other words, these columns show the likelihood of *not* incurring a time-out, and thus reflect how well $\mathcal{T}_{min}$ can be approximated.

Note that since problems are represented by rather abstract feature vectors, several different problems may be represented by the same feature vector. This general difficulty of feature-based methods—which results from features that are not distinctive enough—becomes particularly evident in connection with automated deduction: Tiny variations in a problem specification can hardly be grasped by features other than very specific ones. As a consequence, a problem P may be considered as similar to some other problem P' as to itself. If P and P' are associated with different heuristics H and H', then it is not guaranteed that the best one, namely H, is applied first to solve P (even though P is in DB_I).

Such kind of "confusion" occurred for instance in connection with problem class ROB. There, $\mathcal{T}_V = \mathcal{T}_{min} + T$ because one problem P was confused with some other problem P' for reasons just explained, and heuristic H' associated with P' did not allow DISCOUNT to prove P before time-out. The fixed sequence performed better than variable sequences for ROB, because one of the five heuristics was the only one that could prove all 10 problems, and it was also either the best heuristic or not significantly

Table 2: "Cross-validation": 90% training problems

	variable sequence ($\mathcal{T}_V$)				fixed sequence ($\mathcal{T}_F$)			
Name	best	average	worst	1^{st}	best	average	worst	1^{st}
BOO	469s	541 s	1669 s	99.64%	1494 s	1586 s	5461 s	96.09%
COL	3412 s	5895 s	12483 s	95.29%	10936 s	11154 s	13924 s	84.29%
GRP	1612 s	2069 s	7673 s	99.66%	13802 s	13802 s	13802 s	85.19%
LCL	561 s	705 s	1765 s	99.4%	1775 s	1786 s	2975 s	91.67%
RNG	165 s	349 s	1368 s	99.3%	2136 s	2224 s	8809 s	86.36%
ROB	186 s	805 s	1386 s	90.21%	235 s	235 s	235 s	98.64%
all	7704 s	11674 s	19161 s	97.71%	43913 s	43975 s	44632 s	85.93%

inferior to the best heuristic for solving a ROB-problem. Such a constellation naturally favors a fixed sequence. But except for ROB, table 1 demonstrates that at least for this "recall precision" test variable sequences clearly outperform fixed sequences.[2]

The more interesting case, however, is when not all problems are training problems. Table 2 displays the results when 90% of the respective problems are designated as training problems. Since there is mostly an astronomical number of possibilities to designate 90% of the problems as training problems, we randomly select training problems t times, and then determine best, average, and worst case with respect to these t trials.[3] In our experiments, we set $t = 1000$.

Table 2 again demonstrates that variable sequences outperform fixed sequences. The columns labeled "1^{st}" display the this time average likelihood that the variable respectively fixed sequences do not incur a time-out. Except for ROB-problems (for reasons explained above), the *average* $\mathcal{T}_V$ (with respect to 1000 random trials) is significantly smaller than the *best case* $\mathcal{T}_F$ of a fixed sequence. Moreover, it is often the case—in particular when considering all problems—that the worst case regarding variable sequences is better than the best case of a fixed sequence. Thus, on the one hand, these experiments document that the problems are not uniform in the sense that some rigid pre-set fixed sequence can produce satisfactory results. On the other hand, the experiments show that our straight-forward NNR-based approach does produce satisfactory results, and hence must be considered to be a viable approach to flexibly automating respectively learning the selection of search-guiding heuristics.

We want to emphasize at this point that our experiments were conducted under realistic and intricate conditions involving a generally difficult search problem (automated deduction) and a diverse collection of problems (TPTP) that were rather deceptive than supportive for our approach.

[2]In case a given problem P is as similar to P' as it is to $\hat{P}$, both P' and $\hat{P}$ being in DB_I, we may resolve this ambiguity with the help of the fixed sequence: H' associated with P' is tested first (second) if H' occurs before (after) $\hat{H}$ in the fixed sequence. But we shall not do this here in order to keep variable and fixed sequences strictly separated so as to avoid diluting our experimental studies.

[3]This procedure is closely related to *cross-validation* known from classification (e.g., [MST94]).

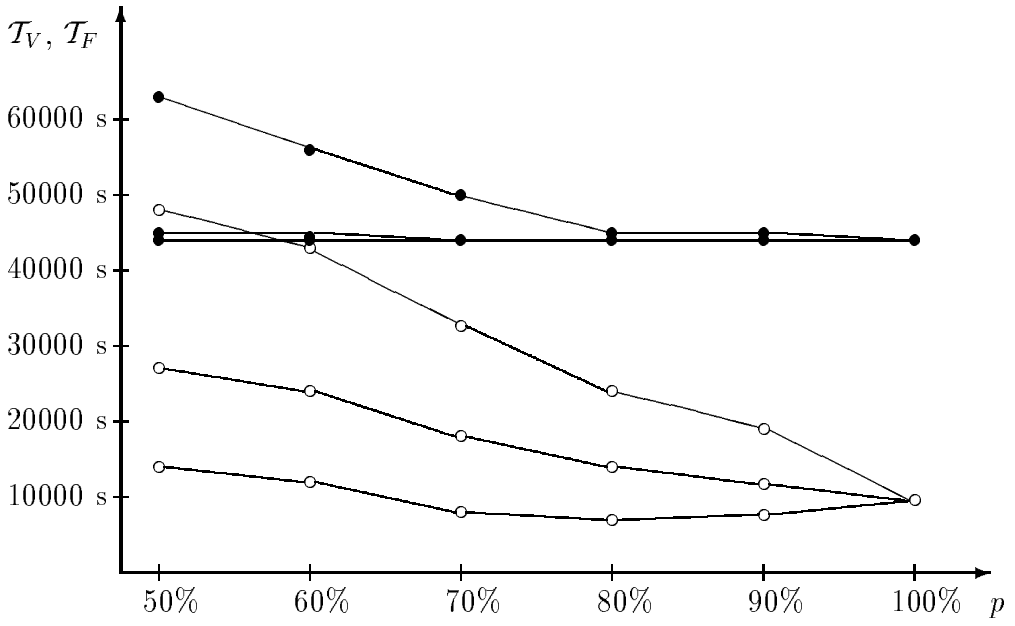

Figure 1: Comparison of best, average, and worst case $\mathcal{T}_V$ (empty circles) and $\mathcal{T}_F$ (filled circles).

Furthermore, we examined how the percentage p of training problems, i.e., the number of problems in DB_I that determine fixed and variable sequences, affects performance. Figure 1 depicts the best, average, and worst case $\mathcal{T}_V$ and $\mathcal{T}_F$ for all problems when using $p = 50\%, 60\%, \ldots, 100\%$. Best, average, and worst case $\mathcal{T}_V$ ($\mathcal{T}_F$) are represented by the respective bottom, middle, and top empty (filled) circle. (Above 60% best and average cases regarding fixed sequences do not differ significantly. Note that for $p = 100\%$ there is only one way to designate training problems—namely all problems are training problems. Hence, best, average, and worst case coincide.)

Figure 1 reveals that in particular worst case $\mathcal{T}_V$ deteriorates as p decreases. This behavior is natural and explicable by the fact that if DB_I contains fewer problems (when p is smaller) then it is possible that a larger part or even all of the training problems in DB_I are "atypical" or exceptions in the sense that they deviate significantly from possible general tendencies and thus are deceptive and misleading.

This observation can also be made in connection with fixed sequences, although fixed sequences are less sensitive regarding a decreasing size (and possibly quality) of DB_I. This is in part surely due to the fact that the performance of fixed sequences is considerably inferior to variable sequences so that there is in a way less room for deterioration. But also, establishing a good fixed sequence does not depend on DB_I as critically as variable sequences do. Therefore, the best case and in parts also the average case $\mathcal{T}_F$ are unaffected by decreasing p down to 50% (cp. figure 1).

Please note that for p ranging from 70% to 90% the best case $\mathcal{T}_V$ is better than the best case $\mathcal{T}_V$ for $p = 100\%$ (which coincides with average and worst case $\mathcal{T}_V$). That is, in a way less information improves performance. The explanation for this observation

is kind of dual with respect to the reason for the worst case $\mathcal{T}_V$ to be affected by far the most when p decreases: When less problems are training problems, it is possible that DB_I contains only "typical" problems that conform with and represent best general tendencies. The deceptive problems are the ones that are omitted. Such a DB_I is better suited than a DB_I that comprises all problems including the deceptive ones. Naturally, if p falls short of some lower boundary (somewhere between 60% and 70% for this experiment), then the best case $\mathcal{T}_V$ also starts to deteriorate, because then the training problems cannot even cover the full spectrum of tendencies, and substantial lack of information rather than deceptive information becomes a dominant factor. All in all, identifying and discriminating "typical" from "atypical" (training) problems obviously can improve performance and thus is an interesting object for further studies. (This possibility has already been investigated in connection with general classification. See, for instance, [Zh92] and [DK95].)

Finally, our experiments have demonstrated that the NNR-based approach produces satisfactory results even for rather small percentages of training problems. (Note that for general classification tasks classifiers are commonly evaluated using 90% of the given data as training data.) Also, it clearly outperformed the rather "naive" and simplistic use of a fixed sequence. Furthermore, we believe that the NNR-based approach can rival human experts. Humans also often employ related (feature-based) criteria to select an appropriate search-guiding heuristic in order to solve a given unknown problem.

4 Discussion

We have presented an approach to automating the selection of search-guiding heuristics employed by a problem solver (based on search). The approach centers on representing problems with vectors of feature values. Similarity between problems is determined with the help of a distance measure on feature vectors. A database of problems, each of which is associated with the heuristic that was employed to solve the respective problem, is used to suggest heuristics to solve a novel problem based on the similarity between problems. In order to cushion possible poor suggestions, it is imperative to operate with a time-out and to suggest a list of alternative heuristics that are tried out one by one—each attempt being limited by the time-out—until the first one succeeds.

Our approach is essentially a slight modification of instance-based learning with the nearest-neighbor rule ([AKA91]). Despite its simplicity, our experimental studies have revealed that it is a viable approach that can cope with hard problem areas like automated deduction. Furthermore, like any approach based on the nearest-neighbor rule, it has simple, yet effective incremental learning capabilities that make it more flexible and powerful than rigid "built-in" approaches. Its performance indicates that it is a useful tool for closing one of the last and major gaps that prevent problem solvers (and in particular automated deduction systems) from being widely accepted or even acceptable, namely to automate the selection of adequate heuristics that are to control the search conducted by a problem solver.

Naturally, limitations of instance-based learning are inherited by our approach. In particular finding an appropriate distance measure, dealing satisfactorily with noise (that

essentially corresponds to "atypical" problems in our case), and superfluous attributes (features) are the most prominent problems for which (partial) remedies already exist (e.g., [AK89], [KD91], [FA96]).

References

[ADF95] **Avenhaus, J.; Denzinger, J.; Fuchs, M.:** *DISCOUNT: A system for distributed equational deduction*, Proc. 6^{th} RTA, Kaiserslautern, GER, 1995, pp. 397–402.

[AK89] **Aha, D.W.; Kibler, D.:** *Noise-Tolerant Instance-Based Learning Algorithms*, Proc. 11^{th} IJCAI, 1989, Detroit, MI, USA, pp. 794–799.

[AKA91] **Aha, D.W.; Kibler, D.; Albert, M.K.:** *Instance-Based Learning Algorithms*, Machine Learning **6**:37–66, 1991.

[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.

[CH67] **Cover, T.M.; Hart, P.E.:** *Nearest Neighbor Pattern Classification*, IEEE Transactions on Information Theory, Vol. IT-13, Jan. 1967, pp. 21–27.

[DK95] **Datta, P.; Kibler, D.:** *Learning Prototypical Concept Descriptions*, Proc. 12^{th} International Conference on Machine Learning, Tahoe City, CA, USA, 1995, pp. 158–166.

[FA96] **Fuchs, M.; Abecker, A.:** *Optimized Nearest-Neighbor Classifiers Using Generated Instances*, Proc. 20^{th} German Conference on AI (KI-96), Dresden, GER, LNAI 1137, 1996, pp. 71–83.

[KD91] **Kelly, J.D.; Davis, L.:** *Hybridizing the genetic algorithm and the k nearest-neighbor classification algorithm*, Proc. 4^{th} International Conference on Genetic Algorithms, San Diego, CA, USA, 1991.

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

[MST94] **Michie, D.; Spiegelhalter, D.J.; Taylor, C.C.:** *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, 1994.

[SSY94] **Sutcliffe, G.; Suttner, C.; Yemenis, T.:** *The TPTP Problem Library*, Proc. CADE-12, Nancy, FRA, 1994, LNAI 814, pp. 252–266.

[Zh92] **Zhang, J.:** *Selecting typical instances in instance-based learning*, Proc. 9^{th} International Conference on Machine Learning, Aberdeen, Scotland, 1992, pp. 470–479.