Explanation-Based Generalization in Game Playing: Quantitative Results

Stefan Schrödl

Institut für Informatik
Albert-Ludwigs-Universität
Am Flughafen 17, D-79110 Freiburg, Germany
e-mail: schroedl@informatik.uni-freiburg.de

Abstract. *Game playing* has attracted researchers in Artificial Intelligence ever since its beginnings. By comparison with human reasoning, learning by operationalization of general knowledge, as formalized by the *Explanation-Based Generalization (EBG)* paradigm, appears to be highly plausible in this domain. Nevertheless, none of the previously published approaches is (provably) sufficient for the target concept, and at the same time applicable to arbitrary game states.
We trace this paradox back to the lack of the expressive means of *Negation as Failure* in traditional EBG, and constructively support our claim by applying the respective extension proposed in [Schr96] to the chess endgame king-rook vs. king-knight.
Methodically, endgames are well-suited for quantitative evaluation and allow to obtain more rigorous results concerning the effects of learning than in other domains. This is due to the fact that the entire problem space is known (and can be generated) in advance.
We present the main results of a large-scale empirical study. The issues of training complexity, speedup for recognition and classification, as well as the question of optimal reasoning under time constraints are analyzed.

Keywords: Explanation-Based Learning, applications

1 Introduction

Explanation-Based Generalization (EBG) is a speedup learning technique which tries to re-organize existing knowledge, guided by a given "typical" problem instance, such that similar instances can be solved more efficiently in the future. A general, domain-independent algorithm was early developed in the context of logic programming [KCMcC87]: Initially, a *domain theory* D (set of facts and rules) is already available which includes a correct and complete definition of the *target concept* (a distinguished atom Q). However, D may be overly general and cumbersome for its intended use by the *performance element* (such as a Prolog system). Thus, the primary aim is to re-formulate D in a more practical form as specified by an *operationality criterion* which defines a list of predicate symbols allowed to occur in the derived rule. In order to cover a frequent usage pattern a *training example* is employed, i.e., a set of facts E such that an instance $Q\sigma$ of Q is logically implied by $D \cup E$. The *explanation* is identified

with the respective *proof*; it is generalized (e.g., by pruning the proof tree at operational atoms) and transformed into a new rule C, which is a logical consequence of D and generalizes E (i.e., $E \cup \{C\} \models \exists(Q\sigma)$ holds).

By means of C, similar instances whose proof structure differs only in operational features can subsequently be recognized more efficiently than using D alone. By changing the inference strategy such that C is always tried before resorting to D (in a Prolog system, it simply suffices to add C at the beginning), it is hoped to improve also the overall performance. However, since in general C is not complete, nothing can be deleted in return. Hence, an additional computational overhead is inevitably imposed for cases where C cannot be successfully applied, particularly for all negative examples. If training instances are not well-chosen, and if numerous rules are learned, the overall efficiency can even degrade; this undesirable effect has been termed the *utility problem* [Mi90].

Numerous factors are involved in a quantitative analysis of the effects of EBG, including specific characteristics of the performance element, and the statistical distribution of the target concept. In order to be predictive, similarly as in PAC-learning, the assumption is made that the distribution of past examples does not significantly differ from that of future ones. Since a mathematical model soon becomes too complicated, the common approach is to address the problem of efficiency empirically [Kel88, Mi90].

Game Playing. We consider two-player, perfect information (the game state is totally accessible to both players), zero-sum (what is good for one player is bad for his opponent) games. The value of an arbitrary legal state (e.g. 1, 0, and -1 for won, drawn, and lost, respectively) can be determined by exploring the respective *game tree* until all its *terminal states* are reached, whose values are fixed by the rules. Search procedures such as *minimax* or *alpha-beta* then back them up by maximizing resp. minimizing the values of successor states on every other level. In the equivalent *negmax* formulation, both players try to maximize their "subjective" evaluation with equal absolute, but opposite sign. Since exhaustive search is mostly intractable, the usual solution is to cut off the game tree at a predefined depth threshold and to propagate a *heuristic evaluation function* instead of the true value.

EBG in Game Playing: Previous Approaches. Game playing has attracted researchers in Artificial Intelligence ever since its beginnings: problems, though well-structured and concisely described, are computationally expensive due to combinatorial complexity; despite this fact, human reasoning is considerably efficient and still unmatched for some games such as Go.

All relevant knowledge (i.e., the rules of the game) can be described in a concise and exact way. Nevertheless, a novice has to become acquainted with typical situations and strategies in order to achieve an acceptable quality of play. Although, in principle, such patterns could be invented from scratch by only reflecting on the rules, it is much easier to inspect examples from actual (commented) matches and to extract the general principles behind them. Indeed, this is the prevalent style of chess textbooks. For example, once we have detected

and memorized the pattern of a pinning (e.g., by search applied to Fig. 2 (a)), we are able to recall it immediately in new situations (as in Fig. 2 (b)), thereby saving computation time. Thus, by comparison with human behavior, exploiting the potential of speedup learning by knowledge operationalization, as formalized by the EBG paradigm, appears to be highly plausible.

In fact, several publications address the issue of Explanation-Based Learning in the domain of two-player games ([FlaDi86, Min84, Ta89, YSUB90] is an incomplete but representative list). In brief, the approach taken in most cases is to represent states and move operators in a STRIPS-like fashion [FiNi71], to use *goal regression* [Wa75] to propagate (winning or losing) conditions along the edges of the game tree back to the root, and to finally conjoin all conditions obtained in this way from different edges. Two main inherent problems are associated with this view. First, the STRIPS-language (essentially consisting of conjunctions of atoms without function symbols) turns out to be too restricted for more complex rules, as e.g. for the variable-length moves of the rook in chess; this results in descriptions which are awkward and not well-suited for generalization. Secondly, in order to be correct, a learned condition has to take into account all the legal moves which could be available at its application time, not only those ones occurring in the training example. Predicting them in advance is a difficult task. Consequently, to the best of our knowledge, all previously reported approaches suffer from significant weaknesses. Minton's Constraint-Based Generalization [Min84], which was implemented for tic-tac-toe, go-moku, and chess, is restricted to certain types of states, where a tactical strategy exists that confronts the losing player at each point with two independent threats which cannot be blocked simultaneously Tadepalli's *Lazy Explanation-Based Learning* [Ta89] intently sacrifices soundness by considering only a single line of play; derived rules may be subsequently refined at the time their use leads to a fault. Generally, the paradox that EBG is not directly applicable, although a complete axiomatization of the domain theory can be provided, was termed the *intractable theory problem.*

2 Overcoming the Difficulties by the Use of Extended EBG for Negation as Failure

The outlined difficulties seem to contradict the fact that a *universal* EBG procedure was already known [KCMcC87]. However, this traditional technique requires the domain theory to be represented as a set of pure *Horn clauses*, whose bodies consist of conjunctions of atoms. We argue that a declarative and natural logical description of game playing concepts requires the expressive means of *negation* (Of course, this is a practical, knowledge engineering requirement, as definite programs are already Turing-equivalent). Informally, excluding draws for simplicity, a game state is won (for the current player) if some legal move exists leading to a successor state that can *not* be won (by the opponent). Thus, a straightforward negmax implementation is along the lines of the clause win(X) $\leftarrow$ move(X, Y), $\sim$win(Y). The more detailed version *negMaxAtLeast* of Fig. 1 holds if the game tree with root *State*, where it is *Player*'s turn to move, truncated at the level of N plies, has a value which is greater than or equals *Bound*. The equivalent minimax formulation depends on negation in a similar way.

```
negMaxAtLeast(State, Player, N, Value) ←
    (terminalState(State, Player); N = 0),
    value(State, Player, Value1),
    Value1 >= Value.
negMaxAtLeast(State, Player, N, Value) ←
    ~terminalState(State, Player),
    N > 0, N1 is N − 1,
    opponent(Player, Opponent),
    Value1 is −Value,
    legalMove(State, State1, Player),
    ~negMaxGreater(State1, Opponent, N1, Value1).
```

Fig. 1. Testing the negmax value of a game state. We assume *negMaxGreater/4* to be analogously defined by replacing ">=" by ">" in the first clause and recursively calling *negMaxAtLeast* in the second one.

The need for negation has partly been obscured by the alternative use of non-logical *set predicates* (such as Prolog's *findall* or *setof*), which can also be found in most Prolog textbooks (Note that set predicates can likewise be implemented in a declarative way using negation, but not via Horn programs alone due to their monotonicity). While their use is justified when efficiency is the primary concern, the lack of a logical semantics excludes them from a general treatment in EBG.

Since negation lies beyond the scope of traditional EBG, its use in game playing was seriously restricted, as revealed by previous approaches. As it shall be demonstrated below, it is now possible to overcome this limitation on the basis of an extension of traditional EBG to the rule of *Negation as Failure*, which was proposed in [Schr96]. Herein, the Clark completion [Cla78] is adopted as a declarative semantics of SLDNF-resolution. Roughly speaking, in contrast to the case of success, explaining failures involves taking into account all possible proof alternatives. Therefore, the main idea of the described method is to generalize the usual concept of *proof trees* (i.e., *AND-trees*) to so-called *AND/OR/NOT-trees*. By this means, success and failure become exactly dual cases. We implemented a Prolog-meta-interpreter which builds, in parallel to the usual derivation, the corresponding AND/OR/NOT-tree by successively replacing atoms by their completed definitions. Operational atoms, as well as those atoms which are not selected anywhere in the derivation, are pruned. The constructed AND/OR/NOT-tree can be interpreted as a first-order expression, which is finally converted into a normal program using a set of simple equivalence transformations. Related ideas how to explain failures can be found in [SiPu88] and [Pu94].

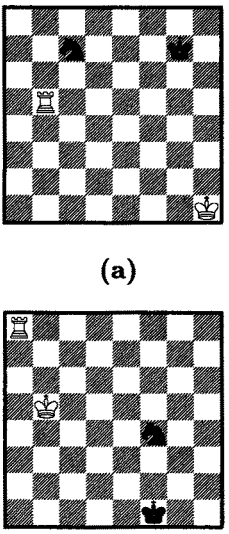
3 The Chess Endgame King-Rook vs. King-Knight

In general, endgames lend themself readily to quantitative evaluation. The complexity of problem instances can be clearly defined and controlled (namely, by the search depth). In contrast to other domains where it might be difficult

to estimate the population of future instances, here the entire problem space is a priori known: it can be completely generated, e.g. through retrograde analysis (as we did in this case, too). This allows us to assess the distribution of positive as well as of negative examples for the target concepts, which is crucial for the prediction of accuracy and efficiency of derived programs. We subsequently keep to the hypothesis that all legal positions are equally likely.

Our testbed was the chess endgame king and rook against king and knight (*krkn*, for short). This choice was inspired by Quinlan's studies with ID3 decision trees [Qu83]. Though not too complex for extensive evaluation, this application exhibits non-trivial features (such as the variable-length moves of the rook).

Suppose w.l.o.g. the rook is white. Consistent with [Qu83], we define black to have immediately lost if he is either checkmate, or if the black knight has been captured and he cannot retaliate by capturing the white rook. A *State* is *won in at most N plies* (for odd N) for a *Player* if it follows from this termination condition by a negmax tree of depth at most N; then *negMaxAtLeast(State, Player, N, 1)* of Fig. 1 holds. Call it *won-N-ply* if additionally it is not won in less than N plies. For even N, *lost-N-ply* is defined analogously (in this case, $\sim$*negMaxGreater*(*State, black, N,* -1) holds). Fig. 2 shows two won-3-ply positions.



(a) **(b)**

Fig. 2. Won-3-ply positions

Out of the approximately 15 million possible ways of placing the four pieces on the board, about 75% are legal black-to-move positions and approximately 64% are legal white-to-move positions (the difference arises because, for instance, the white king cannot be in check in a black-to-move position). Table 1 lists, for some concepts Q (won-N-ply resp. lost-N-ply), the ratio h_Q of positive instances with respect to all legal (white-to-move resp. black-to-move) positions.

3.1 Implementation

In the negmax procedure of Fig. 1, the predicates *legalMove/3*, *terminalState/2*, and *value/3* encapsulate the entire game-specific knowledge. For our purposes, we used a slightly simplified chess theory which borrows from that of Flann and Dietterich [FlaDi89]. A state is described using the predicate *on/3*; e.g., *on*(*state1, (8,1), [king,white]*) holds in Fig. 2 (a). In [FlaDi89] a state consists of sixty-four assertions which list empty squares by a special dummy "piece"; however, this can be simplified by using negation, since we can now define a square to be empty "by default" unless proven otherwise. Predicate *connected/3* specifies adjacent squares together with their direction. Move generation dis-

Table 1. (Notation explained in the text)

depth	concept Q	h_Q[%]	Δt_Q [%]	$\bar{\epsilon}_Q$ [%]	$t^{rel}_{Q,min}$ [%]	t^{rel}_Q [%]	$\epsilon_Q(t+\Delta t)/\epsilon_Q(t)$ [%]
1	won-1-ply	24.54	5.54	18.0	85.6	102 – 123	83.2
2	lost-2-ply	3.495	2.15	56.4	94.9	97 – 119	80.5
3	won-3-ply	6.032	0.87	79.0	95.9	104 – 119	78.4
4	lost-4-ply	0.951	0.39	89.0	98.6	99 – 116	76.8
5	won-5-ply	3.449	0.19	91.3	97.0	102 – 107	63.7

tinguishes between *singleStep* pieces such as king and knight, and *slidingPieces* such as the rook. For each type, the set of move vectors (*legalDirection*) is provided. For the latter class, it has to be checked whether all intermediate squares between the source and the destination are empty. In order for the generalization to cover moves of arbitrary length, not just the length occurring in the example, this property was expressed using two predicates *between/3* and *openline/4*. States derived by move sequences are represented as in situation calculus, with appropriate regression and frame axioms provided.

3.2 Experimental Setting

Let D_{op} denote the definitions of operational predicates, and let D be the negmax procedure of Fig. 1, together with our chess program except for D_{op}. The only relation that changes across training instances is *on/3*; four such assertions determine a *krkn* board position, which are always given in the order white king/white rook/black king/black knight.

Our *operationality criterion* $\mathcal{OP}$ consists of the predicates *on/3*, *connected/3*, *openline/4*, *between/3*, *opponent/2*, *singleStepPiece/1*, *slidingPiece/1*, *legalDirection/2*, as well as of some general arithmetic built-ins.

The chosen target concepts Q were won-N-ply, for $N \in \{1,3,5\}$, and lost-N-ply, for $N \in \{2,4\}$. Our available hardware resources did not allow a comprehensive study beyond this depth. The algorithms proposed in [Schr96] are able to deduce a common generalization for multiple training examples by merging several AND/OR/NOT-trees into one. For each such target concept Q, a sequence of programs $C^Q_1, C^Q_2 \ldots$ is derived from a corresponding sequence of successively larger training sets $\mathcal{E}^Q_1, \mathcal{E}^Q_2, \ldots$, where $\mathcal{E}^Q_{i+1}$ is obtained from $\mathcal{E}^Q_i$ by adding a randomly drawn board configuration E^Q_i which is a positive instance of Q, and which is not yet covered by the previous program C^Q_{i-1} (i.e., the eight coordinates are repeatedly created by a random number generator according to a uniform distribution until these two properties are satisfied). $E^{\text{won-3-ply}}_1$ is depicted in Fig. 2 (a). The proof for Q is always carried out under the Prolog selection rule, applying the basic negmax procedure of Fig. 1 without any heuristics.

We measured Prolog's execution time using the SICSTUS profiling package; by counting the number of calls, backtracks, choice-points, etc., artificial units are calculated in a machine-independent way [GoKe87].

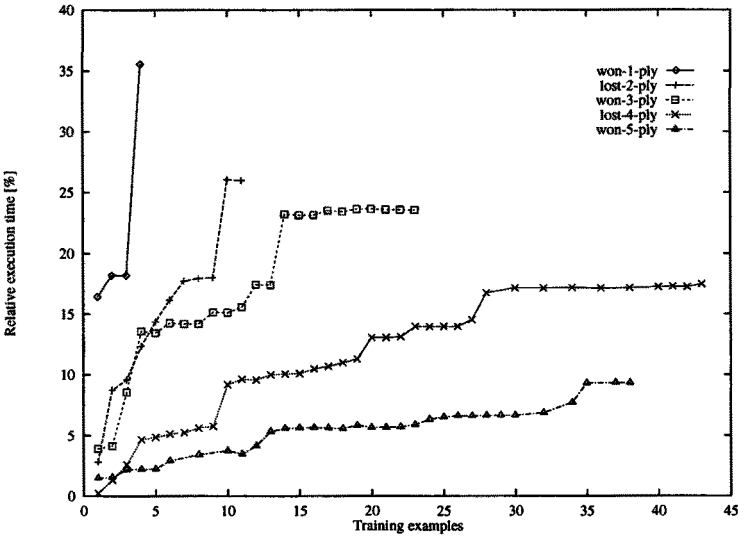


Fig. 3. Utility for positive instances

4 Results

4.1 Speedup for Recognition

First, consider the mere recognition task, where the existence of a solution is known in advance. Each C_i^Q is tested on a random sample of 1000 board positions $\tilde{E}_j^Q$ which are positive instances of Q and do not occur in the training set. We determine the average execution time $t_Q^+(C_i^Q)$ for the attempt to prove Q from $C_i^Q \cup D_{op} \cup \tilde{E}_j^Q$, over all j. The *relative execution time* is obtained by dividing it by the respective figure $t_Q^+(D)$ for the original domain theory. E.g., D needed 124378633 units to process a won-3-ply test set in this way, compared to 4844826 units using the program $C_1^{\text{won-3-ply}}$ derived for the first training example; hence, the respective relative execution time amounts to 3.9%. Fig. 3 plots these values for each program C_i^Q in dependence of the size i of the training set.

For a fixed concept Q, $t_Q^+(C_i^Q)$ grows approximately linear (with slightly decreasing slope) with i. A better speedup can be achieved for greater search depth. The values Δt_Q, defined as the average of $(t_Q^+(C_{i+1}^Q) - t_Q^+(C_i^Q))/t_Q^+(D)$ over all examples, can be found in the fourth column of Table 1.

Since no move sorting is applied, the arrangement of clauses within the program strongly influences the order of exploration, and hence on execution time. In Fig. 2 (a) only the rook takes an active part in white's winning strategy; thus, if its position is asserted first unlike our standard order, execution time for the original domain theory is reduced from 134982 to 100676 units. The derived program $C_1^{\text{won-3-ply}}$, however, is almost indifferent to this permutation (4610 compared to 4618 units). As also a closer look at the generated code reveals, its

search for moves of the winning side is already restricted to the subset of rook moves (more precisely: moves of sliding pieces). We might state that selectivity is the very source of performance gain.

As a very coarse-grained model, assume that a program derived for one training example uniformly considers, on each level of the game tree, only a constant fraction p of the moves taken into account by D. Then we should expect the relative execution time to grow proportionally to p^d, where d is the search depth of the target concept. A corresponding least square fit for the fourth column of Table 1 yields a value of $p = 43\%$.

4.2 Training Complexity

Generally, an algorithm for learning from examples is called efficient if it is capable of producing approximations with tolerable error using only moderate computational resources, essentially time. Since the generalization effort of EBG is proportional to that of proof construction, we may focus on the number of training examples needed to achieve a given accuracy threshold.

Recall that the error of EBG is biased: positive instances may be taken for negative ones, but not vice versa. Therefore, we measure the generality of a derived program C_i^Q by means of a new test set of 1000 randomly drawn positive instances of Q. For each of them, an attempt is made to prove Q using C_i^Q. The number of failures, divided by 1000, yields the ratio $\epsilon(C_i^Q)$ of misclassifications. E.g, the program $C_1^{\text{won-3-ply}}$ deduced for Fig. 2 (a) on 1000 randomly drawn won-3-ply states succeeded in 186 cases, one of which is shown in Fig. 2 (b). The generalization agrees well with our intuitive concept of a pinning, abstracting e.g. from the actual number of intermediate squares on the line passed by the rook. Fig. 4 plots the error $\epsilon(C_i^Q)$ of C_i^Q, in dependence of the training size i.

A general merit of knowledge-based methods such as EBG is their low training complexity, as compared to inductive techniques, where often thousands of instances must be supplied. This is also confirmed by our data: e.g., 90% of all won-3-ply states can be recognized by learning from 6 examples.

However, the required training size grows nearly exponentially with the admissible error. For plausibility, assume that every instance leads to a generalization with the same error $\bar{\epsilon}_Q$, and that all these generalizations are statistically independent. Then we should expect the error after having presented a training set of size n to be approximately proportional to $(\bar{\epsilon}_Q)^n$: a fixed number of additional examples reduces the error by a constant factor. In fact, a least square fit explains our experimental data quite neatly (with correlation between 0.96 and 0.99). Its results are shown in the fifth column of Table 1.

Note that an informed teacher could achieve the same accuracy with fewer examples, since he is able to draw examples deliberately rather than randomly as in our experiments. Ideally, training examples should be chosen to illustrate different concepts so that their overlap is minimized. Then, the training size for exact identification is supposed to be of order $1/(1 - \bar{\epsilon}_Q)$, e.g., about a dozen for a search depth of five plies.

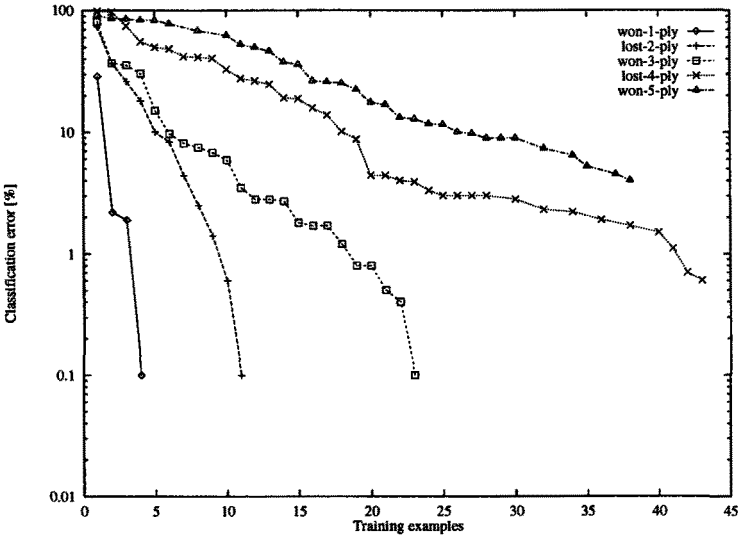


Fig. 4. Training complexity

4.3 Utility for Classification

The utility of a program C has to be judged with respect to the overall task of *classifying* examples as being positive or negative instances of the target concept Q, rather than of merely *recognizing* the positive ones. The effort of a proof attempt crucially depends on whether it succeeds or fails. We have defined $t_Q^+(C)$ above to be the average execution time required to finish a successful proof for a randomly chosen *positive* example of Q using C; similarly, let $t_Q^-(C)$ be the average time until the attempt for a random *negative* example fails. We measured $t_Q^-(C)$ using another test set of 1000 legal (white-to-move resp. black-to-move) *krkn* positions not satisfying Q. Proving a concept won-N-ply can be finished as soon as one successful move at the root has been discovered, whereas in the case of failure, all of them are explored. It turned out that $t_Q^-(C)$ is generally larger (on the average, twice) than $t_Q^+(C)$. The opposite is true for concepts lost-N-ply.

We may estimate the average time $t_Q(D)$ to classify instances w.r.t. Q solely by the original domain theory D as $h_Q \cdot t_Q^+(D) + (1 - h_Q) \cdot t_Q^-(D)$. Suppose that a derived program C is added at the beginning of D; it succeeds in a fraction $h_Q \cdot (1 - \epsilon(C))$ of all cases. Positive examples which are misclassified by C occur with relative frequency $h_Q \cdot \epsilon(C)$; otherwise, the average total classification time is composed of the average failure time of C and that of D. Define

$$\begin{aligned} t_Q(C \cup D) := \quad & h_Q(1 - \epsilon(C)) \cdot t_Q^+(C) + (1 - h_Q(1 - \epsilon(C))) \cdot t_Q^-(C) \\ & + h_Q \epsilon(C) \cdot t_Q^+(D) \qquad + (1 - h_Q) \cdot t_Q^-(D). \end{aligned} \tag{$*$}$$

The *utility* of C can then be read off $t_Q^{rel}(C)$, defined as the ratio $t_Q(C \cup D)/t_Q(D)$ of the execution times with resp. without its addition: $t_Q^{rel}(C) < 1$ indicates an improvement, $t_Q^{rel}(C) > 1$ an overall delay.

In [Schr96], multiple examples are treated such that, as an upper bound for the relative execution time, $t_Q^{rel}(C) \leq 2$ holds. In the limit, the derived program approximates the original domain theory, so that essentially the same steps are carried out at most twice (in the case of negative examples). On the other hand, the assumption $\epsilon(C) = t_Q^+(C) = t_Q^-(C) = 0$ in $(*)$ (i.e., the learned program classifies all positives examples correctly and does not consume any time) yields a lower bound which is independent of the presented training examples and the form of the derived programs. We have $t_Q^{rel}(C) \geq t_{Q,min}^{rel}$ with $1/t_{Q,min}^{rel} = 1 + h_Q/(1 - h_Q) \cdot (t_Q^+(D)/t_Q^-(D))$. Informally, the usefulness of EBG grows with the probability of the target concept, and with the effort of recognizing positive instances using the domain theory, as opposed to the rejection of negative ones. In our case, we obtain the values shown in the sixth column of Table 1.

These bounds already explain the the ranges of actually observed relative execution times, depending on the number of training examples (seventh column). The overall improvement is only minor, in general performance even degrades up to 123 % in the worst case. Since positive examples are quite infrequent, their accelerated recognition can hardly outweigh the necessary delay for negative ones. Similarly as in other domains where EBG is applied, we have to face the utility problem.

However, as an alternative to the usual scheme, we might sacrifice completeness by discarding the original domain theory. Taking each example not successfully recognized by C (possibly erroneously) as a negative one results in *approximate* classification. In this case, $t_Q(C)$ can be estimated by dropping the third and fourth terms from $(*)$. Accuracy can be gradually traded for performance: using more training examples reduces the error rate of a derived program, but at the same time increases the average execution time when it is applied. In this way, we arrive at Fig. 5.

E.g., for the concept lost-4-ply, 17.5% of the execution time used by the original domain theory are sufficient if we agree on a misclassification of 0.6% of the positive instances. The greater the search depth N, the more can be gained. Spending an additional constant amount of computation time (by adjusting the training size) roughly reduces the error rate by a constant factor. Denote the average error reduction gained for one additional percent of the time used by the original domain theory as $\epsilon_Q(t + \Delta t)/\epsilon_Q(t)$; a least square fit yields, with correlation between 0.93 and 0.98, the values of the eighth column of Table 1.

In game playing, reasoning under time bounds is an important issue; such constraints are usually imposed by tournament conditions. We could store an ordered sequence of programs for approximate classification (whose first element is the original domain theory itself), along with execution time estimates; then, decide on-line which one of them to use, based on the actually available time. Informally, such a behavior results in selectively exploring deeper parts of the search tree at interesting points while occasionally "overlooking" unlikely variants, as opposed to directly adjusting the global search depth of the unaltered original program in response to time resources. It appears to be appropriate both from the viewpoints of efficiency and cognitive adequacy.

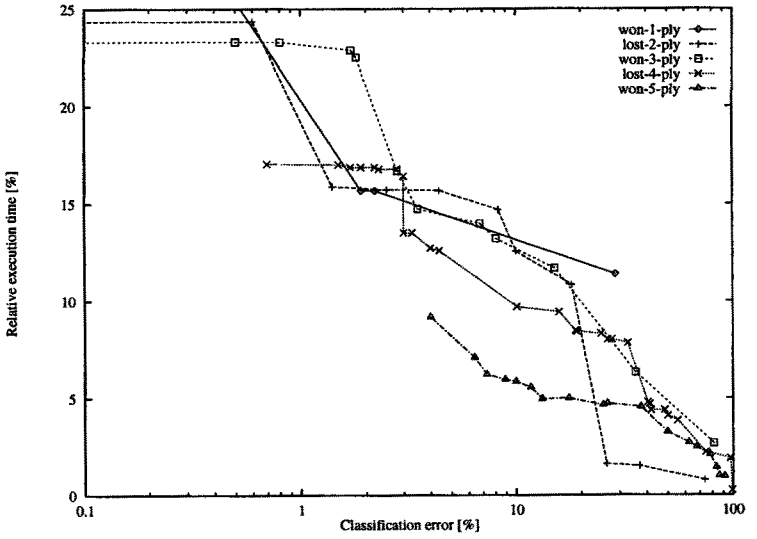


Fig. 5. Performance – accuracy trade-off

5 Conclusion and Discussion

By comparison with humans, Explanation-Based Learning appears to be well-suited for game playing. However, previous approaches failed to provide a universal and provably correct method. We point out that a serious handicap was the fact that the expressive means of Negation as Failure is not considered by the traditional procedure for EBG of [KCMcC87], but that it is nonetheless necessary for a declarative and natural formalization of game tree search. This problem can now be remedied on the basis of the extension proposed in [Schr96].

From a methodical viewpoint, the domain of endgames allows for a more precise and detailed quantitative evaluation than those domains for which similar studies were reported earlier. This work describes the main results of an application and large-scale experimental evaluation of EBG for the chess endgame king-rook vs. king-knight; such an inquiry has not been undertaken before.

A significant speedup can be achieved for recognizing positive examples; the impact of learning strongly grows with the search depth. The number of training examples necessary to achieve a given level of accuracy is strikingly small. Nevertheless, in the overall *classification* task the utility problem occurs, mainly due to the low proportion of the target concepts. Hence, our observations suggest to focus on *approximate* classification instead by discarding the original domain theory. In this case, we can implement *reasoning under time constraints* and nearly continuously trade accuracy for performance by adjusting the training size.

Obviously, our results are only valid with respect to our fixed chess program and for the Prolog selection rule; the influence of different designs of the domain theory and of different inference strategies was not examined.

Quinlan [Qu83] pointed out that most of the effort in *krkn* classification lies in the task of *feature construction*, i.e., finding easily computable attributes of states significant for their outcome. To this end, he applied inductive methods to semi-automatically extract them from a complete endgame database. Interestingly, some of the reported features are very similar to generalizations derived by EBG (such as the shown example of a "pinning"). We believe that combining EBG and similarity-based methods for this task warrants further studies.

References

[Cla78] Clark, K.L., *Negation as Failure*, in: Gallaire, H., and Minker, J. (eds.), Logic and Data Bases, Plenum Press, NY, (1978), pp 293-322

[FiNi71] Fikes, R. E., and Nilsson, N. J., *STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence 2(3-4), pp 189 -208

[FlaDi86] Flann, N.S., and Dietterich, T.G., *Selecting Appropriate Representations for Learning from Examples*, AAAI-86, (1986), pp 460-466

[FlaDi89] Flann, N.S., and Dietterich, T.G., *A Study of Explanation-Based Methods for Inductive Learning*, Machine Learning 4, Kluwer Academic Publishers, Boston, (1989), pp 187-226

[GoKe87] Gorlick, M.M., and Kesselman, C.F., *Timing Prolog Programs Without Clocks*, Symp. on Logic Programming, IEEE Computer Society, (1987), pp 426-432

[KCMcC87] Kedar-Cabelli, S.T., and McCarty, L.T., *Explanation-Based Generalization as Theorem Proving*, Proc. of the Fourth International Workshop on Machine Learning, Irvine, (1987)

[Kel88] Keller, R. M., *Defining Operationality for Explanation-Based Learning*, Artificial Intelligence 35, (1988), pp 227-241

[Min84] Minton, S., *Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples*, AAAI-84, (1984), pp 251-254

[Mi90] Minton, S., *Quantitative Results Concerning the Utility of Explanation-Based Learning*, Artificial Intelligence 42, Elsevier Publishers, (1990), pp 363–391

[MKKC86] Mitchell, T., Keller, R., and Kedar-Cabelli, S., *Explanation-Based Generalization: A Unifying View*, Machine Learning 1:1, (1986), pp 47-80

[Pu94] Puget, J.-F., *Explicit Representation of Concept Negation*, Machine Learning 14, Kluwer Academic Publishers, Boston, (1994), pp 233–247

[Qu83] Quinlan, J. R., *Learning Efficient Classification Procedures and their Application to Chess End Games*, in: Michalski, et al. (eds.), Machine Learning: An Artificial Intelligence Approach, San Mateo, CA, Morgan Kaufmann, (1983), pp 463–482

[Schr96] Schrödl, S., *Explanation-Based Generalization for Negation as Failure and Multiple Examples*, ECAI-96, Budapest, Hungary, John Wiley & Sons, (1996), pp 448–452 (1986)

[SiPu88] Siqueira, J. L., and Puget, J. F., *Explanation-Based Generalization of Failures*, Proc. of ECAI-88, Munich, (1988), pp 339–344

[Ta89] Tadepalli, P., *Lazy Explanation-Based Learning: A Solution to The Intractable Theory Problem*, IJCAI-89, Detroit MI, (1989), 694-700

[Wa75] Waldinger, R., *Achieving Several Goals Simultaneously*; in: Elcock, E. W., and Michie, D. (eds.), Machine Intelligence 8, Ellis Horwood, Chichester, England, pp 94-138

[YSUB90] Yee, R.C., Saxena, S., Utgoff, P.E., Barto, A.G., *Explaining Temporal Differences to Create Useful Concepts for Evaluating States*, AAAI-90, (1990), pp 882-888