Constructive Induction On Decision Trees

Christopher J. Matheus* and Larry A. Rendell
Inductive Learning Group, Computer Science Department
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana, Illinois 61801

Abstract

Selective induction techniques perform poorly when the features are inappropriate for the target concept. One solution is to have the learning system construct new features automatically; unfortunately feature construction is a difficult and poorly understood problem. In this paper we present a definition of *feature construction* in concept learning, and offer a framework for its study based on four aspects: detection, selection, generalization, and evaluation. This framework is used in the analysis of existing learning systems and as the basis for the design of a new system, CITRE. CITRE performs feature construction using decision trees and simple domain knowledge as constructive biases. Initial results on a set of spatial-dependent problems suggest the importance of domain knowledge and feature generalization, i.e., *constructive induction.*

1 Introduction

Good representations are often crucial for solving difficult problems in AI. Finding suitable problem representations, however, can be difficult and time consuming. This is especially true in machine learning: learning can be relatively easy if the training examples are presented in a suitable form, but when the features used in describing examples are inappropriate for the target concept, learning can be difficult or impossible using selective induction methods. To overcome this problem a learning system needs to be capable of generating appropriate features for new situations.

This paper is concerned with the automated construction of new features to facilitate concept learning, an issue closely related to the "problem of new terms" [Dietterich *et* a/., 1982] and *constructive induction* [Michalski, 1983]. We begin by defining "feature construction" in the context of concept learning from examples, and then proceed to identify four inherent aspects: *detection, selection, generalization,* and *evaluation.* These aspects comprise an analytical framework for studying feature construction which we describe through examples drawn from the existing systems of BACON.1, BOGART, DUCE, PLSO, and STAGGER. This framework serves as the basis for the design of CITRE, a system that performs feature construction on decision trees using simple domain knowledge. The results of our initial experiments demonstrate CITRE'S ability to improve learning through feature construction in a tic-tac-toe classification problem. Extensions of CITRE for this and other problems are also discussed.

*This research was funded by a University of Illinois Cognitive Science/Artificial Intelligence Fellowship and ONR Grant N00014-88K-0124.

2 The Problem

We state the following definition of feature construction[1] in concept learning:

> Feature Construction: the application of a set of constructive operators $\{o_1, 0_2, \ldots o_n\}$ to a set of existing features $\{f_1, f_2, \ldots . f_m\}$) resulting in the construction of one or more new features $\{f_1, f_2, .F_N\}$ intended for use in describing the target concept.

This definition emphasizes the notion of a constructive operator, defined as a function mapping a tuple of existing features into a new feature. A constructive operand is a tuple of features to which a constructive operator is applied. For convenience, an operator/operand pair will be referred to as a constructor, up until the time it becomes a new feature. The constructor *and(big(i),red(i)),* for example, consists of the two place operator *and(binary, binary)* and the operand tuple *(big(i),red(i)).*

Additional implications of this definition include: 1) A feature is a function mapping instances into values. 2) All new features are defined in terms of existing features, such that no inherently new information is added through feature construction. 3) The definition can be applied iteratively: after a new feature is created, it may serve as part of an operand in a subsequent round of construction. 4) A separate, selective induction algorithm is assumed to make use of the constructed features in attempting to describe the target concept.

[1] Feature construction is often referred to as "constructive induction." We, however, reserve the term constructive induction to refer to the prediction of unobserved, disjunctive regions in instance space (see section 3.3).

3 Four Inherent Aspects

Representative examples of systems that perform feature construction include FRINGE [Pagallo, 1989], BACON [Langley *et* a/., 1987], STAGGER [Schlimmer, 1987], DUCE [Muggleton, 1987], PLSO [Rendell, 1985] and BOGART [Newman and Uhr, 1965]. These systems employ a variety of techniques to a wide range of learning problems, making it difficult to identify exactly when, where, and how feature construction is performed. Through our effort to understand feature construction in such systems we have discovered some common issues and concerns. In particular, we have identified the following four aspects believed to be inherent to the problem of feature construction:

1. detection of when construction is required
2. selection of constructors
3. generalization of selected constructors
4. evaluation of the new features.

These four aspects are not necessarily present in every system performing feature construction. Nor do they necessarily delineate sharply defined phases of the feature construction process. Instead, they represent four identifiable issues inherent to the general problem of feature construction that account for much of the perceived variability between systems. As a result, these aspects have proven useful as a framework for the analysis and comparison of systems (see [Matheus, 1989]), as well as in the development of CITRE. We now consider each aspect in detail.

3.1 Detecting the Need

If the original feature set is sufficient for the selective induction algorithm to acquire the target concept, feature construction is unnecessary. Because the constructive process can be computationally expensive, it is often desirable to determine *when* feature construction is needed. There are at least three general approaches to determining when to perform construction:

- no detection *(i.e.,* always perform construction)
- analysis of the initial data set
- analysis of a concept description.

Some systems have no detection mechanism because they continually perform construction. The sole purpose of BACON.l, for example, is to construct new features. A system might base detection on an analysis of the initial training set, for example using cluster or factor analysis (although none of the observed systems fully develop this approach). More typically, systems perform detection by observing the results of the selective induction system, *i.e.,* the concept description it produces. Feature construction may be deemed necessary if the learning system fails to produce a description, or if the description fails to satisfy some measurable quality *(e.g.,* accuracy, conciseness, comprehensibility). Using failure to trigger construction is employed, for example, by STAGGER [Schlimmer, 1987], The presence of excessive disjuncts in a concept description can also be used for detection, as suggested in [Rendell, 1988]. We return to this last approach in section 4 to discuss its use in CITRE.

3.2 Selecting Constructors

After determining the need for feature construction, a system must select one or more constructors to use in the creation of new feature(s). The difficulty with this selection process is that the space of potential constructors is generally intractably large. Even in the simple case of a problem represented by N Boolean features the number of potential constructors is 2^2 . For more complex problems, such as those involving numerical features, the search space can be infinite. The task of selection is to pick out a small subset of these potential constructors satisfying the representational requirements of the current problem. This selection process can be analyzed in terms of two phases: initial and runtime selection.

The first step in reducing the constructor space is to reduce the set of *potential* operators to a more manageable size. This *initial selection* has been approached in two ways. Either a small set of simple, domain-independent operators is chosen *(e.g.,* {AND, OR, NOT}), or a set of problem- or domain-specific operators is developed *(e.g.,* "counting the number of wheels on a box car"). Simple operators have the advantage of being applicable to a wide range of problems, but they require multiple iterations of the constructive process in order to build up complex new features *(e.g.,* STAGGER, PLSO). Problem-specific operators can reduce the complexity of the constructive process and thereby decrease construction time, but at the expense of being limited in application and also requiring effort and knowledge for their development.

The reduced set of constructive operators obtained from initial selection represents a system's "candidate" operator set. In every system we have encountered, the "candidate" operator set is significantly smaller than the "potential" set. Even so, this set is still intractably large in most cases, and further selection is required at runtime to choose which operators to apply, and which operands to apply them to. This *runtime selection* amounts to ordering the set of candidate constructors. We have observed four approaches to ordering candidate constructor sets related to the following biases:

- algorithm biases
- training set biases
- concept-based biases
- domain-based biases.

Algorithm biases occur when the implementation of a feature construction algorithm arbitrarily selects one constructor over another. These biases are typically undesirable, being unjustified by the data or the model. Some systems use information in the training set instances to help guide selection. STAGGER takes this approach by using instances that fail to agree with the concept as the basis for suggesting new features. Likewise, in B0GART individual instances are used as templates for new features. Similarly, concept descriptions can be used to bias constructor selection; this approach can be especially useful if a concept description is close to the target concept (we show how this approach can be used on decision trees in section 4). Domain knowledge can also

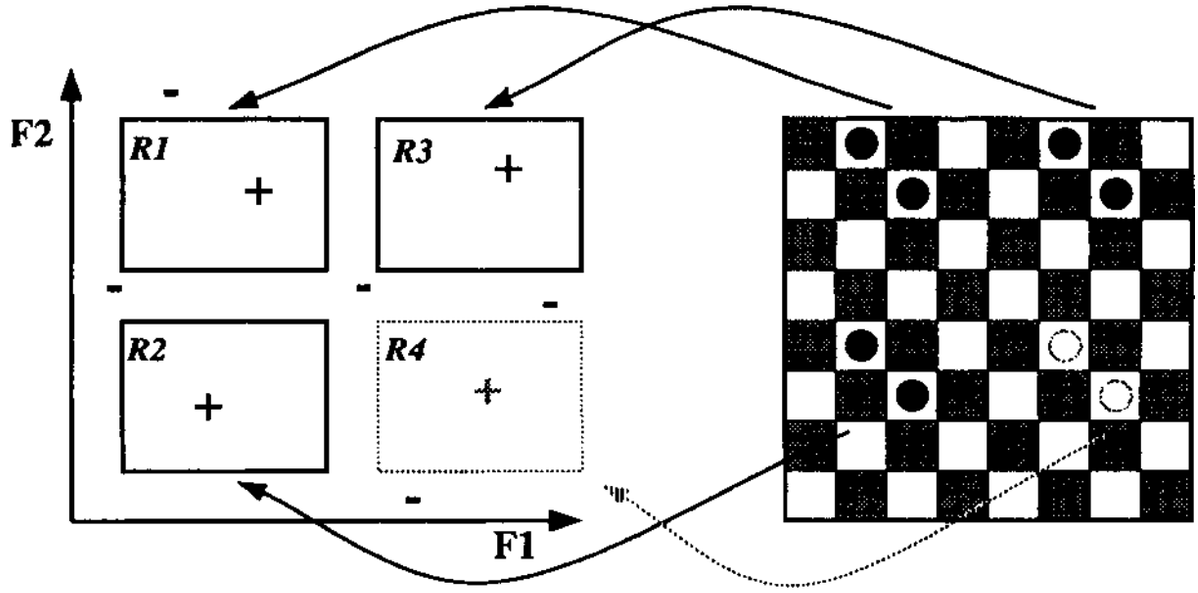

Figure 1: Constructive Compilation versus Constructive Induction. If a system, upon observing instances from R1, R2, and R3, constructs the new feature *or(Rl, R2, R3)* it is performing *constructive compilation.* If a system generalizes this feature into *or(Rl,R2,R3,R4),* thereby predicting the existence of an unobserved region of instances (the white pieces) in R4, it is performing *constructive induction.*

serve as a selection bias, either as a set of heuristics for selecting useful constructors, or as a set of filters for rejecting undesirable candidate features. Our experiments with CITRE suggest that even simple domain knowledge can be a powerful selection bias (section 4).

3.3 Generalizing Constructors

The set of selected constructors may be highly specific to a set of instances or a concept depending upon the selection biases employed. Specific constructors can be generalized in ways analogous to how descriptions are generalized in learning concepts, *e.g.,* changing constants to variables, dropping terms (see [Michalski, 1983]). If a feature construction algorithm does not generalize beyond its input data, then it is only able to "compile" observed features and feature values into new terms. We refer to this form of feature construction as constructive compilation. If generalization is performed during feature construction, the features generated may "predict" regions of positive and negative instances that have not been observed. We call this form of feature construction constructive induction.

Figure 1 depicts a situation in which observed pairs of pieces on a checkers board (the dark pieces) have been mapped into disjunctive regions in an instance space of two abstract features. A constructive algorithm that merges these three regions, *Rl, R2, and R3,* into a new feature *or(Rl, R2, R3),* is performing *constructive compilation.* If on the other hand, the algorithm generalizes this new feature to *or(Rl, R2, R3, R4),* thereby predicting the unobserved pair of pieces (the white pieces), the algorithm is performing *constructive induction.*

For certain learning algorithms, constructive compilation alone can significantly improve learning. Consider a system that describes concepts by selecting one feature at a time *(e.g.,* a decision tree inducer). Problems can arise when the target concept depends on the correlation of two or more features, because a feature that splits the data poorly by itself may provide a very good split when paired with another feature. If a constructor is able to find this pair and compile it into a new feature, the construction of a more accurate and/or concise concept becomes possible. This result is observed in our experiments with CITRE (see also [Pagallo, 1989]).

Constructive induction is potentially more powerful than constructive compilation because it can produce new features with far fewer observed instances (an important quality when the training set is sparse). The problem with constructive induction is that it requires strong biases in order to produce valid generalizations. Strong and appropriate generalization biases are most readily available in the form of domain knowledge. The strong bias for the generalization *or(Rl, R2, R3, R4)* in Figure 1, for example, might have come from knowledge that in checkers a feature useful at one board location can often be translated to other board locations. As our preliminary results suggest, the use of relevant domain knowledge can significantly affect the quality of constructed features (section 4).

3.4 Evaluating Features

If the number of new features grows too large, it may be necessary to evaluate and discard some. The evaluation of new features can be approached in at least three ways:

- ignore the problem (keep all features)
- request evaluation from the user
- order the features and keep the best ones.

The first approach *(e.g.,* in BACON.1, PLSO, and BOGART) is the simplest but most limited in that it is only appropriate if the number of new features remains relatively small. The second approach *(e.g.,* in DUCE) places

the burden on the user who must be sufficiently knowledgeable to judge the quality of new features. The third approach *(e.g.,* in STAGGER and CITRe) is autonomous, but it requires a measurement of feature "quality" (i.e., the credit assignment problem). One solution is to use the measure employed for selecting features during concept formation as the measure for new feature evaluation; we consider this technique further in the next section. Other measures of feature evaluation have been considered (see [Seshu *et* a/., 1989]).

4 CITRE

The framework defined by these four aspects was used in the development of CITRB, a new system for performing constructive induction on decision trees using simple domain knowledge.[3] CITRE is similar to FRINGE [Pagallo, 1989] in its use of learned decision trees to suggest useful constructors. Both systems iterate between tree learning and feature construction: a decision tree is learned, the tree is used to construct new features, the new features are used in the creation of a new tree, and the process is repreated until no new features are created. CITRE differs in its use of domain knowledge, feature generalization, and feature evaluation.

As suggested above, the key to tractable and effective feature construction is the use of strong biases to help the system converge to an appropriate set of new features. CITRE uses three primary biases: concept-based operand selection, domain-knowledge constructor pruning, and an information-theoretic evaluation measure. These biases, depicted in Figure 2, will now be described in terms of how CITRE handles each of the four aspects in the context of a tic-tac-toe classification problem *{i.e.,* classifying boards as winning or losing instances).

Detection: Feature construction is performed any time disjunctive regions are detected in a candidate decision tree, as evidenced by the presence of more than one positively labeled terminal node.

Selection: The initial selection of operators was based on two criteria: First, we wanted to minimize the amount of domain-specific information present in the constructive operators. Second, we wanted to test the hypothesis that useful, higher-level features can be constructed incrementally through the iterative application of *simple* constructive operators. We consequently selected the simple, generic, binary operator *and(binary, binary).* The *and(~, ~)* operator used in conjunction with the negation implicit in decision trees provides complete representational capability for new features.

The primitive features used in the tic-tac-toe problem are the contents of the nine board positions: *F* — {posII, posl2, posl3, pos21, pos22, pos23, pos31, pos32, pos33}. These are nominal features having values x, o, or blank. In order to accommodate the binary operand

[2] CITRE currently operates in conjunction with COGENT [Matheus, 1988], a decision tree induction program functionally comparable to ID3 [Quinlan, 1986]. Both CITRE and COGENT are implemented in Quintus Prolog running on Sun3 workstations.

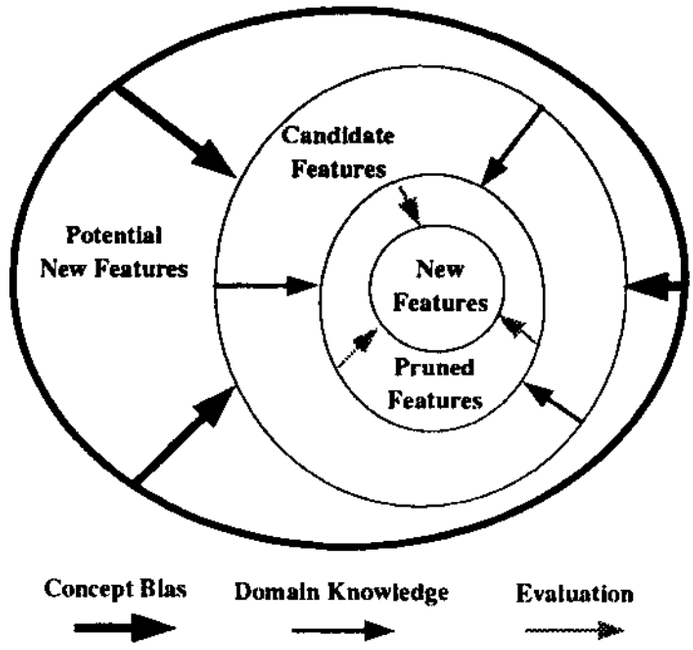

Figure 2: CITRE's Main Biases. The large set of potential new features is significantly reduced by 1) concept-based selection biases, 2) simple domain-knowledge pruning, and 3) feature evaluation based on an information theoretic measure.

slots of the *and(_,_)* operator these nominal features are converted into boolean expressions as necessary during runtime selection *(e.g., equal(posll,x)).* The size of the class of potential operands in this case (9 features, 3 values) is thus 27 * 26 = 702. If N new features are actually created in round one, then round two has *(N* + 27) **(N* + 26) = N^2 + 537V + 702 potential new features; the number of new features will similarly increase for all subsequent rounds.

The main bias used by CITRE in selecting the appropriate operand is concept based, and is derived from the idea of "merging disjunctive regions" [Rendell, 1988]. Disjunctive regions in the decision tree are described by the branches leading from the root to the positively labeled terminal nodes. CITRE collects the feature-value pairs *(e.g.,* posII=x) at nodes along these positive branches and proposes all pairs of these binary feature-value pairs *(e.g., equals(pos 11,x))* as candidate operands. In a complete tree of depth D, this method results in 2^{D-l} candidate operands. In tic-tac-toe where D is typically less than 6, this bias results in fewer than 35 operands being proposed out of greater than 700 potential operands. This large reduction (> 700 to < 35) gives a good indication of how strong this bias is (as implied by the heavy arrows in Figure 2). For each operand selected a new candidate feature is constructed by applying the *and()* operator *(e.g., and(equalf pos 11, x),equal(pos 12, x))).*

Domain knowledge can serve as an additional bias to filter out less promising candidate features (see Figure 2). In CITRE, domain knowledge is limited to simple facts (ground literal clauses) defining permissible relationships between constructive operands. For the tic-

tac-toe problem we have implemented a small amount of general knowledge about board games: knowledge about adjacency of pieces being important *(i.e.,* only consider new features having adjacent constituent features), and knowledge about the significance of piece type *(i.e.,* only consider new features composed of the same type pieces). When CITRE uses this domain knowledge, the constructor *and(equal(posll,x),equal(po$32, o))* would be rejected on both grounds.

Generalisation: In the experiments described in this paper we used the single generalization operator "changing constants to variables." If a constructor's operand consists of two features having the same value, a generalized feature is proposed with its values replaced by a variable. For example, the generalization of *and(equal(poall, x), equal(po$12, x))* becomes *and(equal(po8ll, VARIABLE), equal(posl2, VARIABLE))*. Both the original candidate feature and its generalization are added to the set of features. When a candidate feature is generalized, the resulting new feature is nominal, rather than binary, and its domain is (x,o,true). For a particular instance, the value of a generalized feature is false if the logical *and()* relationship does not hold, or the variable's value otherwise *(i.e.,* x or o).

Evaluation: Although CITRE may generate hundreds of constructors while working on a given problem, it only keeps a maximum of 27 (9 primitive + 18 constructed) features at a time. Its criterion for evaluation of features is the same as that used in deciding which feature to select during tree formation, *i.e.,* an information theoretic measure. The "utility" of each feature is measured in terms of the information gained by using the feature to split the entire training set into disjoint subsets. They are then ordered by utility, and those features (excluding the primitives) with the lowest utilities are deleted until the total feature count is again down to 27. As shown in Figure 2, this evaluation is the final bias used in pruning the set of potential new features.

Evaluating features by their ability to effectively split the current data set works well in CITRE because the feature with the highest utility necessarily becomes the first node in the subsequent decision tree. This greedy approach can fail, however, if a new feature having a poor utility on the entire data set exhibits a relatively high utility sometime after the first split. For this reason, other forms of "deeper" evaluation are being considered.

4.1 Experimental Results

We conducted four series of tests in which we varied the use of generalization and domain knowledge as shown in Table 1. All four series were run on an identical collection of fifteen data sets. Each data set consisted of 100 randomly-generated tic-tac-toe instances, labeled either as "win for x" or "win for o." Each run iterated between tree construction and feature construction until no new features could be constructed. All decision trees constructed during a test were analyzed for accuracy on classifying a test set of 200 randomly chosen boards.

Table 1 summarizes the results for all four test series. Columns two and three indicate the use of domain knowledge and generalization for each test. The column labeled "First" lists the accuracies for the original decision trees averaged over the 15 data sets. Corresponding accuracies for the final decision trees are in the column labeled "Last." The difference between the these two columns appears in the "Difference" column. The ± values indicate the 95 percent confidence intervals as determined by a t-test. Under "New Terms" are the average number of new features considered per data set (measured after selection and generalization but before evaluation), with the highest number in any single run shown in parentheses. In the last column is the average difference between the number of nodes in the original trees and the number in the final trees.

On average, feature construction resulted in improved classification accuracy. For the last three tests - those making use of domain knowledge and/or generalization - this result is significant with 95 percent confidence, with an average improvement of around six percent. Whereas there is no significant difference among the accuracy improvements of these three tests, the use of generalization resulted in a greater variance in accuracy. This observation reflects the fact that although generalization on average improves performance, it can lead to decreased performance on individual runs when invalid generalizations are made.

Another difference between the use of generalization and domain knowledge is evident in the number of new features considered. Without domain knowledge an average of over 360 new features were considered per data set, with a single worst case occurring in the generalization test with the production of 941 candidate features. With the addition of the domain knowledge described above, the average drops below 40, and in the worst case is still less then 60. This difference translates directly into a significant improvement in efficiency.

In terms of number of internal decision tree nodes, the average difference does not change significantly between tests. Because the number of nodes in the trees remains constant while the accuracy is improving the difference can be attributed to the use of better features when domain knowledge or generalization is used.

4.2 Extensions and Other Applications

In addition to tic-tac-toe, CITRE has been applied to the problem of learning random boolean functions. Even on the simple problems thus far tested *(e.g.,* 4 term, 3 features/term DNF functions), accuracy improvements of greater than ten percent have been observed. Detailed experiments in this domain are proving useful in the analysis of several aspects of CITRE's algorithm, including variations on the concept-bias, enhanced operator sets, and the effect of various evaluation methods.

Our preliminary work on tic-tac-toe is being extended in several ways: 1) larger training sets are being used to demonstrate that generalizations tend to be more valid when based on larger samples of the instance space, 2) new operators are being explored *(e.g., or(_,_), and(_,_,_))* 3) additional domain knowledge regarding board games is being used to help focus the feature construction *(e.g.,* emphasizing corners, straight lines,

Table 1: Tic-Tac-Toe Classification Test Result Averages										
	Biases		Classification Accuracy				New Terms		Node Count	
Test	Gen	DK	First	Last		Difference	Ave.	(Max)	Difference	
Test1			69.8 ± 1.8	71.2 ± 3.3		1.4 ± 3.3	367	(596)	12.1	± 5.2
Test2	X		69.8 ± 1.8	76.7 ± 2.7		6.9 ± 2.5	387	(941)	14.3	± 8.3
Test3		X	69.8 ± 1.8	75.7 ± 1.8		5.9 ± 2.0	27	(40)	10.2	± 3.9
Test4	X	X	69.8 ± 1.8	76.2 ± 4.0		6.4 ± 4.5	37	(57)	11.4	± 6.6

etc.), and 4) more powerful generalization rules are being tested including reflection, translation, and rotation generalizes.

We intend to apply the results obtained from the tic-tac-toe problem to more difficult board game problems such as knight-fork and chess endgames. Towards this end, domain knowledge and generalization rules are being developed specifically for the spatial properties of board games. Our ultimate goal is to develop a more general approach appropriate for the larger class of spatial problems that includes, for example, LED classification, the blocks world, protein folding, etc.

5 Conclusion

The four aspects of feature construction presented here have proven useful as a framework for analyzing existing systems and guiding the development of new constructive techniques. In particular, this framework guided the design of CITRE, a new system that performs feature construction on decision trees using limited domain knowledge and simple generalization. Results on a board game classification problem suggest the appropriateness of CITRE'S three strong biases: concept-based selection, domain-knowledge pruning, and an information theoretic evaluation measure. More specifically, our results suggest the potential importance of domain knowledge and generalization for effective feature construction - two areas to which current feature construction systems have paid little attention (see [Matheus, 1989]).

Acknowledgment

We would like to thank Gunnar Blix, Gregg Gunsch, Carl Kadie, Doug Medin, and David Wilkins for their helpful discussion and comments on the issues developed in this paper.

References

[Dietterich *et al,* 1982] Thomas G. Dietterich, Bob London, Kenneth Clarkson, and R. Geoff Dromey. Learning and inductive inference. Technical Report STAN-CS-82-913, Computer Science Department, Stanford University, 1982. (Also in Ch. XIV of The Handbook of Artificial Intelligence, Cohen & Feigenbaum (Ed.) Morgan Kaufmann, 1982).

[Langley *et aL,* 1987] Pat Langley, Herbert A. Simon, Gary L. Bradshaw, and Jan M. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge, 1987.

[Matheus, 1988] Christopher J. Matheus. COGENT: A system for the automatic generation of code from examples. Unpublished manuscript, 1988.

[Matheus, 1989] Christopher J. Matheus. A constructive induction framework. In *Proceedings of the International Workshop on Machine Learning,* Ithaca, New York, 1989.

[Michalski, 1983] Ryszard S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence,* 20(2):111-161, 1983. (Reprinted in Machine Learning: An Artificial Intelligence Approach, Tioga Press, 1983).

[Muggleton, 1987] Steve Muggleton. DUCE, an oracle based approach to constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 287-292, 1987.

[Newman and Uhr, 1965] C. Newman and Leonard Uhr. BOGART: A discovery and induction program for games. In *Proceedings of the Twentieth National Conference of the ACM,* pages 176-185, New York, 1965. Lewis Winner.

[Pagallo, 1989] Giulia Pagallo. Learning DNF by decision trees. In *Proceedings of the International Joint Conference on Artificial Intelligence,* Detroit, MI, August 1989. Morgan Kaufmann Publishers, Inc.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning,* 1(1), 1986.

[Rendell, 1985] Larry A. Rendell. Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 650-658, August 1985.

[Rendell, 1988] Larry A. Rendell. Learning hard concepts. In *Proceedings of the Third European Working Session on Learning,* pages 177-200, 1988.

[Schlimmer, 1987] Jeffery C. Schlimmer. Incremental adjustment of representations in learning. In *Proceedings of the International Workshop on Machine Learning,* pages 79-90, Irvine, CA, June 1987. Morgan Kaufmann Publishers, Inc.

[Seshu *et al.,* 1989] Raj Seshu, Larry A. Rendell, and Dave Tcheng. Managing constructive induction using optimization and test incorporation. In *Proceedings of the International Conference on Artificial Intelligence Applications,* pages 191-197, Miami, FL, March 1989.