Building Robust Learning Systems by Combining Induction and Optimization

David Tcheng* Computer Science
Bruce Lambert Speech Communication
Stephen C-Y Lu Mechanical Engineering
Larry Rendell Computer Science
University of Illinois
Urbana IL 61801
tcheng@rn.cs.uiuc.edu

Abstract

Each concept description language and search strategy has an inherent inductive bias, a preference for some hypotheses over others. No single inductive bias performs optimally on all problems. This paper describes a system that couples induction with optimization to carry out an efficient search of large regions of inductive bias space. Experimental results are reported demonstrating the system's capacity to choose optimal biases even for complex and noisy problems.

1 Introduction

Research on methods for learning concepts from examples occupies a central position in the discipline of machine learning (Michalski, Carbonell, & Mitchell, 1983). Among those who study the problem of learning from examples, it is now widely recognized that each concept description language and search strategy has an inherent inductive bias. Mitchell (1980) defined *inductive bias* as "any basis for choosing one generalization over another, other than strict consistency with the observed training instances." In our research, we give the term *inductive bias* a functional definition. Given a set of *examples,* an inductive bias is a function that produces a hypothesis, mapping points from a space of input representations to points in a space of output representations. In this paper we focus on inductive systems that produce functions that map a set of *input values* to a single *output function value.* A partial list of inductive systems conforming to this restriction includes: neural nets (Rumelhart & McClelland, 1986), regresssion (Box, Hunter, & Hunter, 1978), decision-trees (Rendell 1983; Quinlan 1983; Breiman, Friedman, Olshen, & Stone, 1984), logic-based approaches (Michalski, 1983), and exemplar-based approaches (Smith & Medin, 1981).

The best inductive bias for a given problem depends heavily on the user's objectives (i.e., hypothesis credibility metric). For instance, if the predominant objective is to produce a hypothesis that is easy to comprehend, a good inductive bias might represent hypotheses as English sentences or graphic representations. In other cases, when predictive accuracy is the primary concern, neural nets, decision trees, and mathematical equations may be preferable. Given the diversity of user objectives, it is clear that no single inductive bias can be optimal for all situations.

To address this problem, our research seeks to produce a robust inductive system in accordance with the methodology first described in Rendell, Seshu, and Tcheng's (1987) discussion of the Variable-Bias Management System (VBMS). The original VBMS paper laid out three methods for developing robust learning systems: integration, optimization, and meta-learning. Integration involves the representation of unique inductive biases as points in a multi-dimensional space (inductive bias space). Optimization is a method for searching inductive bias space. Meta-learning is a method for selecting an appropriate bias optimization strategy based on characteristics of the current problem.

In this paper we present the results of tangible progress towards integration and optimization. To integrate existing inductive biases, we developed the Competitive Relation Learner (CRL), a system that manages a set of diverse inductive biases to produce hybrid concept representations. To optimize inductive bias, we developed an optimization algorithm called the Induce and Select Optimizer (ISO) and applied it to the problem of optimizing CRL's inductive bias.

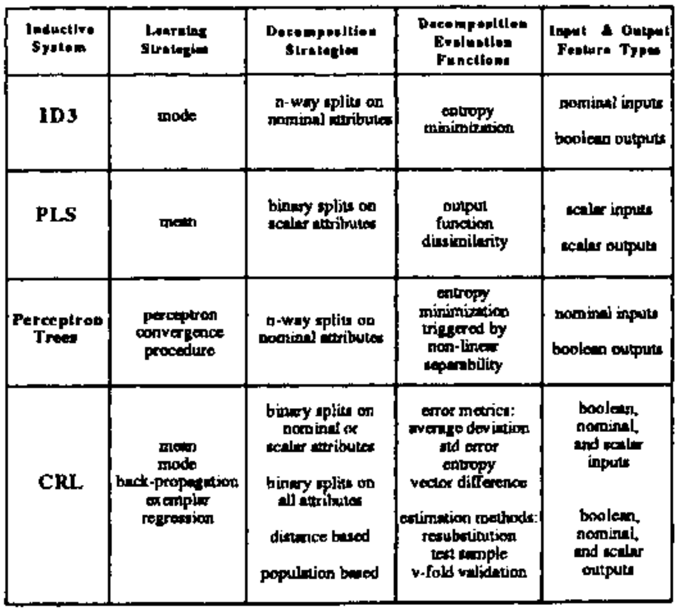
In what follows, we first suggest that well known decision-tree building algorithms such as ID3 (Quinlan, 1986), PLSI (Rendell, 1983), and CART (Breiman *et al.,* 1984), as well as recently developed hybrids like Utgoff s perception trees (1988), can all be viewed as partial instantiations of an abstract class of algorithms we call *recursive splitting algorithms.* Second, we describe CRL, a *generalized* recursive splitting algorithm (Tcheng, Lambert, & Lu, 1989) that provides a framework for integrating multiple methods for function approximation (learning strategies), multiple methods for breaking problems into subproblems (decomposition strategies), and multiple methods for selecting the best set of subproblems to solve (decomposition evaluation functions). Next we argue that optimization is the appropriate methodology for searching CRL's bias space. After a general discussion of optimization, we describe the details of ISO. Finally, we present experimental results that illustrate the capabilities of both CRL and ISO. In concluding, we outline plans for applying the combined system to a large database of real-world learning problems and for learning meta-level

* This research was sponsored in part by the National Science Foundation (DMC-8657116) and by the Applied Intelligent Systems Group of Digital Equipment Corporation.

rules for selecting inductive bias optimization strategies.

2 The CRL System

The principal factor motivating the design of CRL was the observation that the behavior of a recursive splitting algorithm depends on three factors: (1) how predictions are made within regions of input space; (2) how candidate decompositions are generated; and (3) how candidate decompositions are evaluated (see also Breiman *et al.*, 1984). Analysis of traditional recursive splitting algorithms reveals that each method possesses only one learning strategy (i.e., a method for making predictions in each subregion), one decomposition strategy, and one decomposition evaluation function. For example, ID3 (Quinlan, 1983) creates n-way splits on nominal feature dimensions, selects the split that minimizes its entropy function, and assigns the most frequently occurring class label to each subregion (i.e., leaf node). PLS1 creates binary splits perpendicular to scalar feature dimensions, chooses the decomposition that maximizes the difference in the output functions, and attaches the mean output value to each subregion. Utgoff s (1988) novel contribution lay in realizing that performance could be improved by putting more powerful predictors at the leaf nodes. His perceptron-tree algorithm first attempts to classify all instances with a perceptron (i.e., a network of threshold logic units). Failing that, it imposes n-way splits along nominal attribute dimensions, selects the split that minimizes its entropy function, and inserts perceptrons at the leaf nodes. Figure 1 summarizes how these algorithms and CRL instantiate the three crucial component processes: prediction, decomposition, and evaluation.



Inductive System	Learning Strategies	Decomposition Strategies	Decomposition Evaluation Functions	Input & Output Feature Types
ID3	mode	n-way splits on nominal attributes	entropy minimization	nominal inputs boolean outputs
PLS	mean	binary splits on scalar attributes	output function dissimilarity	scalar inputs scalar outputs
Perceptron Trees	perceptron convergence procedure	n-way splits on nominal attributes	entropy minimization triggered by non-linear separability	nominal inputs boolean outputs
CRL	mean mode back-propagation exemplar regression	binary splits on nominal or scalar attributes binary splits on all attributes distance based population based	error metrics: average deviation std error entropy vector difference estimation methods: resubstitution test sample v-fold validation	boolean, nominal, and scalar inputs boolean, nominal, and scalar outputs

Figure 1. Characteristic components of four recursive splitting algorithms.

Any fixed combination of prediction, decomposition, and evaluation strategies may be ideal for a particular class of problems, but will fail to provide optimal performance on others. The CRL system was designed to perform well across a wide range of problems. This robust performance is made possible by CRL's ability to manage multiple, competing component strategies. Such competition often results in the formation of hybrid concepts that simultaneously capitalize on the strengths and minimize the weaknesses of two or more distinct inductive biases (Schlimmer, 1987; Utgoff, 1988). The current implementation of CRL contains multiple learning strategies, multiple decomposition strategies, and multiple decomposition evaluation functions. The design is modular, and new strategies can be added incrementally as long as they adhere to CRL's standard input-output specifications. Below are relevant details of CRL's component strategies.

2.1 Learning Strategies

In traditional recursive splitting algorithms, predictions are made by traversing the decision tree with a given input example and then simply returning the mean or mode of the output points in the specified subregion. Hypotheses generated by such algorithms take the form of discontinuous step-functions. In contrast, were regression or a neural network used at the leaf nodes, at least two advantages would be reaped. The resulting hypothesis would provide a closer fit to continuous functions and fewer decompositions would be necessary. Expanding on this idea, CRL competitively applies a variety of learning strategies (i.e., inductive biases) at the leaf nodes to produce a hypothesis for each subregion. In addition to averaging, CRL can be made to fit subregions with several different models (i.e., statistical regression, neural nets, averaging, and exemplar based strategies). Competition provides the basis for choosing the learning strategy for each region that will be used to form the final hypothesis. The availability of several learning strategies allows CRL to solve complex problems involving multiple models, and it is one of the novel aspects of the CRL system. Below are the learning strategies currently implemented in CRL.

2.1.1 Mean and Mode

These strategies take either the mean or mode of all observed output vectors and return a constant hypothesis. Due to their limitations, the mean and mode learning strategies are almost always combined with one or more decomposition strategies.

2.1.2 Exemplar

The exemplar learning strategy is based on a psychological model of human concept acquisition (Smith & Medin, 1981). In the learning phase, the exemplar strategy randomly selects and remembers a specified number of examples from the training set. In the performance phase, predictions are made by looking through all of the memorized examples and finding the set of n-examples closest to the new example in input space (based on normalized euclidean distance). The average of the *n* closest output points is returned as the predicted output value.

2.1.3 Regression Models

These learning strategies are based on the classical statistical regression model (Box, Hunter, & Hunter, 1978). CRL currently contains the linear, quadratic, logarithmic, and exponential regression models.

2.1.4 Neural Net Models

The last of CRL's component learning strategies is a neural network. Neural networks are highly interconnected assemblies of simple computing elements which, when properly trained, can learn arbitrary input-output mappings. The particular system in CRL is a flexible model designed to let the user explore a wide range of architectures, connectivity patterns, and settings of other parameters. The network learns by the back-propagation of error signals (Rumelhart & McClelland, 1986).

2.2 Decomposition Strategies

The general idea of problem decomposition is of fundamental importance in problem solving (Newell & Simon, 1972). Equipped with a representation of the problem space and operators for moving around in that space, the problem solver's objective is to break down the given problem into subproblems whose solutions can be achieved by applying the available operators. In CRL, the operators are learning strategies and the difference to be reduced is the error of the overall hypothesis.

Mathematically, a decomposition is a function that maps every point in the parent region of input space to one subregion indexed by a subproblem number. For example, if II and 12 are input features, Equation (1) represents a simple binary decomposition function of the sort generated by PLS1.

(1) D(I1, I2) = if (I1 < 3),
then Subproblem 1
else Subproblem 2

In general, a decomposition function may be defined over any or all of the input feature dimensions and may map examples to two or more subregions. In traditional recursive splitting algorithms, only one algorithm for generating decomposition functions (i.e., a decomposition strategy) is available. Because many decomposition strategies exist, CRL allows the user to specify a set of them to use in parallel. Below are the decomposition strategies currently defined in CRL.

2.2.1 Distance Decomposition Strategy

This decomposition strategy is a slight variation on that used by PLS1. First, the minimal hyper-rectangle that contains all the points in input space is calculated. Then a set of candidate decompositions is generated by splitting each input feature dimension at n evenly spaced points.

2.2.2 Population Decomposition Strategy

The major disadvantage of the distance decomposition strategy is that it is insensitive to the actual distribution of examples in input space. In the worst case, many of the decompositions generated will divide input space into two regions, one containing all but one of the examples and the other containing a single example. To avoid this problem, we have developed a method for generating decompositions which partitions input space into regions based on population density. The result is that more densely populated regions of input space undergo proportionately more decomposition (Kadie, 1988).

2.2.3 Hyperplane Decomposition Strategy

All of the previously mentioned decomposition strategies insert region boundaries perpendicular to attribute dimensions. Instead of a single attribute test, this strategy generates arbitrarily placed hyperplane decomposition functions (see Breiman *et a/..*, 1984). The position of the hyperplane is controlled by a parameter which determines how many randomly selected input points will be used to compute the location of an origin. If this parameter is set to ∞, all hyper-planes pass through the centroid of the input space vectors and the population tends to be divided equally. If this parameter is set to 1, the hyperplanes pass through a single, randomly selected point (not necessarily the center), yielding a greater variety of decompositions. Once the origin has been chosen, the orientation of the hyperplane is randomly determined.

2.3 Decomposition Evaulation Functions

Recursive splitting algorithms, such as ID3, typically rely on mathematical measures of a subregion's entropy to evaluate decompositions. Decomposition evaluation functions of this sort favor splits that partition the input space into minimally entropic regions. The logic behind such a strategy (based on information theory) predicts that regions of low entropy will be easier to learn. As such, entropy measures are indirect measures of the overall hypothesis error reduction that a given decomposition will bring about. In contrast to the entropy-based heuristic, the CRL approach to decomposition evaluation is to measure directly the overall error reduction of a candidate decomposition by actually evaluating competing learning strategies in the newly created subregions (see Fig. 2). CRL's decomposition evaluation functions consist of two components, a hypothesis error metric and a error validation strategy.

2.3.1 Hypothesis Error Metrics

A CRL hypothesis error metric is a function taking the actual example and predicted output vector as input and returns a value indicating the error of the prediction. General purpose hypothesis error metrics include average deviation, standard error, entropy, and vector difference. The user can also define domain-specific error metrics. For example, when CRL is used to create diagnostic rules, the relative cost of false positives and false negatives can be built into the error metric and thus used to bias decomposition accordingly. This advantage is unavailable to systems that use only general error metrics because such metrics are insensitive to the type of misclassifications that may result from a given split.

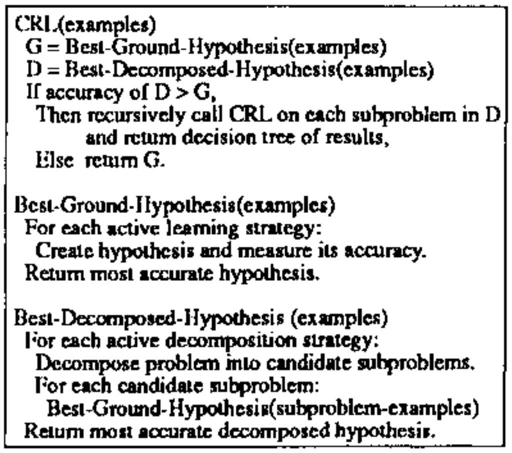
2.3.2 Error Validation Strategies

Whereas an error metric measures the error of a prediction based on a single example, a validation strategy uses the error metric to estimate the average hypothesis error across across an entire example set. CRL currently possesses three error validation strategies, resubstitution, test-sample, and v-fold cross validation (Breiman *et al.,* 1984). Resubstitution tests a hypothesis on the same examples that were used to create the hypothesis. Test-sample requires the user to divide the available examples into training and testing sets. Training examples are used to form the hypothesis, and testing examples are used to estimate the error of the hypothesis. V-fold cross validation is a method for estimating the error of a hypothesis, where the number of folds is the number of groups to partition the examples into. For example, if the number of folds is 10, then the examples are randomly partitioned into 10 equal-sized groups. Next, the examples from all but 1 of the subgroups are used to create a hypothesis with the given learning strategy. The accuracy of the hypothesis is then estimated using the unseen group of examples as test cases. This process is repeated for each group of examples, and the average hypothesis error is calculated. When more than one learning strategy is competing, v-fold cross validation prevents learning strategies with a high degree of freedom from unjustly dominating (e.g., by memorizing all the examples).

2.4 The CRL Algorithm

Figure 2 shows how CRL takes a given set of active learning and decomposition strategies and decides which will be used to form the final hypothesis. This method is a straightforward generalization of the simple recursive splitting algorithm — the difference being that CRL uses a best-first search (without backtracking) strategy to evaluate multiple learning and decomposition strategy combinations in parallel. CRL begins with a single input space region containing every example and estimates the error in the region. The error of a region is determined by applying each active learning strategy to the examples and recording the error of the most accurate hypothesis.

Next, the algorithm determines whether further decomposition will reduce the overall hypothesis error. To do this, CRL applies all active decomposition strategies and evaluates the resulting candidate decompositions by computing the error of the resulting regions in the manner described above. The most valuable decomposition, the one that brings about the greatest overall error reduction, is used to create new subregions. This process is recursively applied to each subregion until one of the following three stopping criteria is met: (1) the error of the overall hypothesis ceases to decrease more than a specified threshold; (2) the number of examples in a candidate subregion falls below a specified threshold; or (3) the time consumed exceeds a specified threshold.



```
CRL(examples)
  G = Best-Ground-Hypothesis(examples)
  D = Best-Decomposed-Hypothesis(examples)
  If accuracy of D > G,
    Then recursively call CRL on each subproblem in D
         and return decision tree of results,
    Else return G.

Best-Ground-Hypothesis(examples)
  For each active learning strategy:
    Create hypothesis and measure its accuracy.
  Return most accurate hypothesis.

Best-Decomposed-Hypothesis (examples)
  For each active decomposition strategy:
    Decompose problem into candidate subproblems.
    For each candidate subproblem:
      Best-Ground-Hypothesis(subproblem-examples)
  Return most accurate decomposed hypothesis.
```

Figure 2. Pseudo code for the CRL algorithm.

From the user's perspective, the main advantage of CRL is that it creates an environment in which one can easily experiment with a diverse set of learning strategies without having to recode data or jump from system to system. Unfortunately, the diversity of choices has a cost. With so many alternative inductive biases to choose from, finding the best bias for a problem is difficult One way around the problem of bias selection is to activate a large, representative set of CRL's learning and decomposition strategies and to let them compete. This approach can be immensely expensive - especially if the user wants to ensure that all significantly different biases are tried. We do not advocate such a brute force solution. Instead, an independent optimization system can be used to search for the inductive bias that is optimal with respect to a given set of user objectives.

To say that optimization should be used to search inductive bias space is insufficient because as many optimization biases as inductive biases exist. Therefore, to complete the definition of our methodology, we must decide on a framework for optimization. In the following section, we describe both weak optimization methods, such as random search, and strong methods, which themselves employ inductive biases to guide the search for the optimum. Rather than choosing between weak and strong methods, we eventually propose a methodology that allows us access to both.

3. Relating Optimization and Induction

3.1 What is Optimization?

From the perspective of decision making, optimization is the process of finding the best decision among a range of untested alternatives (Buchanan, 1986). Optimization problems are defined in terms of a *decision space* and a means for evaluating candidate decisions (the *evaluator,* see Fig. 3).

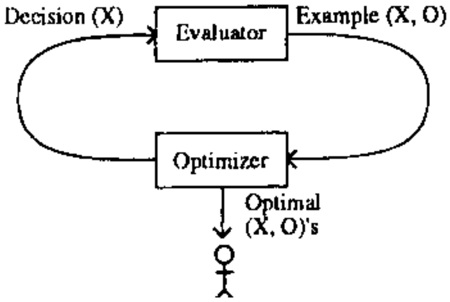


Figure 3. Simple view of optimization.

For example, if the goal is to design an aircraft wing that produces maximal lift, the dimensions of the decision space are attributes of the wing design (e.g., material type, wing curvature, wing length, etc.). In this case, candidate decisions can be evaluated either by building and testing the wing or by estimating the wing performance through computer simulation. The goal of the optimization process is to find the best point in decision space using the least effort.

An optimizer takes as input a set of examples and employs some heuristic to generate new candidate decisions. A continuum of optimization biases ranges from the *weak* to the *strong.* Random selection of candidate decisions is a weak heuristic which can be quite effective when the cost of evaluating decision points is negligible. If the cost of evaluation is high, stronger (and less efficient) selection heuristics are justified. Strong selection heuristics employ some inductive bias to create a hypothesis that describes the objective surface over decision space (see Fig. 5). This hypothesis is used to guide further selection.

Response surface fitting (Box *et al,* 1978) is an example of a strong optimization strategy which uses an inductive bias to aid in the selection of new candidate decisions. Response surface fitting typically uses polynomial regression to estimate the relationship between decision variables and objective score. To generate new candidate decisions, the selector component of a response surface fitting algorithm first calculates the decision point maximizing its polynomial hypothesis and then selects that point as the next decision to evaluate. For each selected point, an objective score is calculated by the evaluation function and associated with the decision space point to form an example for the next iteration of polynomial regression. The process continues until some stopping criterion is met (e.g., the objective score ceases to increase or resources are exhausted).

3.2 The Induce and Select Optimizer

Recognizing that a versatile optimization system ought to posess both the strong and weak methods described above, we designed the Induce and Select Optimizer (ISO). The ISO framework is schematically represented in Figure 4.

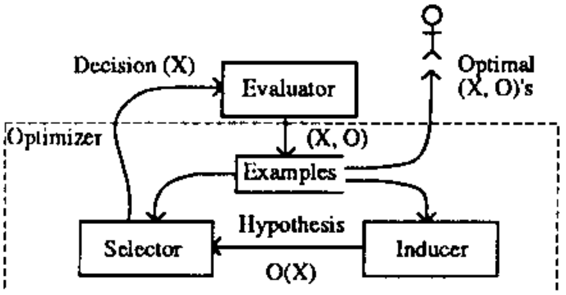


Figure 4. Component view of optimization.

As its name suggests, ISO includes two main components: an *inducer* and a *selector.* The role of the inducer is to describe, for the selector's benefit, the objective surface over decision space (see Fig. 5). Optimization strategies such as response surface fitting use induction, but they are equipped with only one inductive bias (e.g., quadratic regression). This is fine if the objective surface over decision space happens to be similar to a quadratic function. If it is not, however, using the induced quadratic hypothesis to guide the selection of new decisions adds little benefit, and may actually impede progress toward the optimum. ISO escapes this limitation because its inducer manages a collection of competing inductive biases. With many inductive biases to choose from, the inducer within ISO has a much higher probability of accurately describing the objective surface over decision space. ISO is, therefore, much more likely to find a strong optimization method.

The second component of ISO is its selector. Selection is the process of using both the examples and the induced hypothesis to guide the selection of new candidate decisions. In ISO, selection is based on two control parameters called *novelty* and *performance.* A high novelty setting causes ISO to prefer points in decision space that are maximally distant from those already attempted. If novelty were the only consideration, ISO would ignore the induced hypothesis describing the objective surface over decision space, and instead perform random, non redundant search. If performance were the only consideration, ISO would attend only to the induced hypothesis, choosing new candidate decisions that maximize that hypothesis. In this way, the novelty and performance parameters allow ISO to exhibit both strong and weak optimization biases.

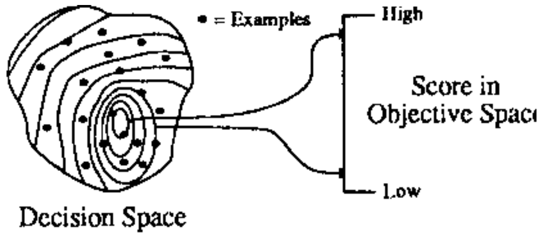


Figure 5. Hypothesis predicting the objective surface over a decision space.

4 Optimizing Inductive Bias With ISO

The availability in CRL of multiple decomposition strategies, learning strategies, and decomposition evaluation functions increases the size of the inductive bias space through which ISO may search. Here we encounter a classic tradeoff. Larger search-spaces are more likely to contain better solutions, but they are also more difficult to search. In this section we describe how ISO optimizes hypothesis credibility over CRL's inductive bias space.

Conceptually, CRL's bias space is a feature space defined by a set of variables that jointly specify which learning and decomposition strategies to use, their control parameters, the hypothesis error metric, how the hypothesis error is to be measured, and the minimum error reduction needed to justify a decomposition. Simply put, a point in bias space completely determines CRL's hypothesis formation behavior.

By selecting which component processes are eligible to be considered, the experimenter defines the region of CRL's inductive bias space in which ISO can seek an optimum. In the first stage of optimization, ISO probes randomly in bias space. Each probe (X) is evaluated by forming a hypothesis with the prescibed inductive bias. The bias point (X) is associated with the achieved hypothesis credibility (O) to form an example. These examples are fed to the induction component of ISO. The inducer within the optimizer outputs a hypothesis (O(X)) that describes the credibility surface over bias space. The selector component of the optimizer uses O(X) and the existing examples to select the next point in bias space to evaluate. This process is the same as that described in Figures 4 and 5, but the decision space is CRL's inductive bias space, and the objective surface is defined in terms of hypothesis credibility (e.g., accuracy, evalution cost, formation cost, comprehensibility, etc.).

5 Experimental Results

CRL's task in this example is to predict the surface roughness of a machined part based on the control parameters of the cutting tool and on the dimensions of the work piece. Examples were generated by a mechanistic simulator for the turning process (Boothroyd, 1975). The simulator mapped four input variables — feed rate (F), depth of cut (D), nose radius (N), and work piece diameter (W) to one output variable — surface roughness (S). Noise was added to the examples so they would more closely approximate real world observations.

For this problem, the user's objective was defined in terms of two factors: hypothesis accuracy (in terms of the variance between predicted and actual outputs) and hypothesis formation time. Accuracy was measured by training ISO on 200 examples and testing on 1000 different examples. For each trial, both training and testing examples were randomly selected. Hypothesis formation time was controlled by an ISO control parameter that placed an upper limit on the amount of CPU time that could be used to form any single hypothesis. For the results reported below, the time limit was 600 CPU seconds (on a SUN/3 180 with 24 Meg).

Figure 6 shows the two best CRL hypotheses produced during the optimization process. Figure 7 shows CRL's performance improvement over time with a 95% confidence interval for the mean hypothesis error superimposed. The minimum possible error was 100 because of the amount of noise added to the examples. The actual function used by the turning simulator to predict surface roughness is given in Equation (2).

$$(2) \qquad S = \frac{0.032 \quad (1000F)^2}{N}$$

Note that only 2 of the 4 input variables were actually arguments to the target function, and that CRL did not possess a learning strategy that could adequately solve the problem without decomposition.

The Best CRL Hypothesis...

```
IF (105F - 10.2D - 31.1N - 0.17W) > -1.30
 THEN
   IF (248F  + 7.4D - 46.5N - 0.23W) > 0.12
     THEN  S = 24500F + 20.1D - 4000N - 2.48W - 4.20
     ELSE  S = 15800F - 56.1D - 1670N - 0.73W  - 4.11
 ELSE
   S = 12400F - 10.6D - 1730N - 0.35W - 22.24
```

Error = 103.5

The Runner Up...

```
IF N < 0.03
 THEN  S = e^(277F - 0.03D - 54.8N + 0.00W + 3.52)
 ELSE  S = 13300F - 53.6D - 1000N + 0.26W -9.82
```

Error = 103.7

Figure 6. The two best hypotheses produced by CRL.

To form a basis for comparison, the problem was also attempted with PLS1, a neural network, and linear regression, where each of these strategies received optimization roughly equivalent (in CPU seconds) to that received by CRL. The accuracies of the best hypotheses from each method are summarized in Figure 8.

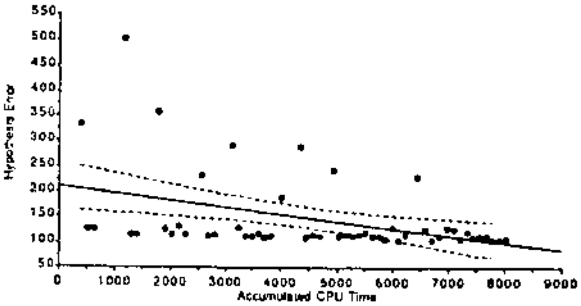


Figure 7. CRL's learning curve for the machine tool prediction problem.

	CRL	Linear Regression	PLS1	Back-Propagation
Accuracy	103.5	121.6	125.9	132.6

Figure 8. Comparision of accuracy of hypotheses produced by CRL and traditional inductive biases.

6 Conclusions and Future Research

The work presented here reflects initial progress toward our ultimate goal: an inductive system which takes as input a problem description (consisting of examples and the user's objectives), returns the hypothesis which is optimal with respect to those objectives, and improves its performance over time by learning from problem solving experience. There is more work to be done before such a system is a reality.

Obvious extensions to the existing implementation include the addition of more component strategies. We plan to add logic-based learning strategies such as AQ (Michalski, 1983), more sophisticated decomposition strategies (that go beyond binary splits), improved selection strategies for ISO (e.g., genetic optimizers (Holland, 1975) and simple hill climbing), and facilities for feature selection and construction.

Beyond adding component strategies to the existing framework, the framework itself needs to be expanded before the system will learn from its own experience. For example, we have shown how an optimizer can be used to find a good inductive bias for a particular problem. However, optimizers themselves have biases such as the method for generating an initial candidate decision to evaluate. ISO begins the optimization process by generating a candidate decision randomly. The system would be more efficient, however, if it possessed meta-level knowledge relating problem characteristics (e.g., number of examples, number of features, type of features, problem domain, maximum hypothesis formation time, etc.) to good points in inductive bias space at which to begin optimization.

To gain this added efficiency, we are currently in the process of implementing the third component of the VBMS framework: meta-learning. Meta-knowledge takes the form of hypotheses that relate problem characteristics to optimal points in inductive bias space (or at least good points at which to begin further optimization). There are at least two approaches to learning this relationship: (1) The system could take problem-description/optimal-bias pairs (saved from past experience) as examples and do induction as usual to learn Best-Bias(Problem). To select a bias point for a particular problem, the system just evaluates this function with the current problem description as its argument; or (2) The system could take problem/bias/objective-score triples (saved from optimization experience) and induce the function describing the objective surface over problem/bias space [i.e., Objective(Problem, Bias)]. To use this function to select the best starting point in inductive bias space, one replaces the problem argument with the current problem description and optimizes Objective(Bias) in the usual manner. Eventually, we plan to test these meta-learning strategies on a large database of real-world machine learning problems (Aha, 1989).

In closing, it is important to emphasize that this particular implementation is just the beginning of a much larger project. It reflects, for the most part, the work of a small group of researchers working primarily on engineering problems. Achieving the three design goals of VBMS (integration, optimization, and meta-learning [Rendell *et al,* 1987]) was relatively easy in such a small, focused group. The challenge over the long term, however, is to achieve these design goals at the level of entire scientific communities. In this paper, we have taken some small steps toward that end.

Acknowledgements

Special thanks to Dr. Guangming Zhang for help in running the turning simulator. Thanks also to David Lambert and Dr. Barbara O'Keefc for helpful comments on an earlier draft. Funding for this research was provided in part by the Applied Intelligent Systems Group of Digital Equipment Corporation, by the National Science Foundation (DMC-8657116).

References

Aha, D. (1989). UCI repository of machine learning domains. Dept. of Computer Science, University of California at Irvine, Irvine: CA.

Boothroyd, G. (1975). *Fundamentals of metal machining and machine tools.* New York: Scripta.

Box, G., Hunter, W., & Hunter, J. (1978). *Statistics forexperimenters.* New York: Wiley.

Breiman, L., Friedman, J., Olshen, R.A., & Stone, C. J. (1984). *Classification and regression trees.* Belmont, CA: Wadsworth.

Buchanan, T. (1986). Multiple objective mathematical programming: A review. *New Zealand Operational Research,* 14: l,(pp. 1-27).

Holland, J. (1975). Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press.

Kadie, C. (1988). Diffy-S: Learning Robot Operator Schemata from Examples. *Proc. of the Sixth International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann. (pp. 430-436).

Lu, S. C-Y., & Chen, K. (1987). A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision making. *Journal of Artificial Intelligence for Engineering Design, Analysis, and Manufacturing,* 1:2, (pp. 109-118).

Michalski, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The AQ inductive learning system: An overview and experiments. Technical report ISG 86-20, Dept. of Computer Science, University of Illinois.

Michalski, R., Carbonell, J., & Mitchell, T. (Eds.). (1983). *Machine learning: an artificial intelligence approach.* Palo Alto, CA: Tioga Publishing.

Mitchell, T. (1980). The need for bias in learning general izations. Technical report CBM-TR-117. Dept. of Computer Science, Rutgers University.

Newell, A. & Simon, H. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Quinlan, R. (1983). Induction of decision trees. *Machine Learning.* 1:1, (pp. 81-106).

Rendell, L. (1983). A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence.* 20:4, (pp. 369-392).

Rendell, L., Seshu, R., & Tcheng, D. (1987). Layere concept learning and dynamically-variable bias management. *Proc.IJCAI '87.* (pp. 308-314). Cambridge, MA: Morgan Kaufmann.

Rumelhart, D., & Mc Clelland, J. (Eds.). (1986). *Parallel distributed processing*, Vol. 1. Cambridge, MA: MIT Press.

Schlimmer, J. (1987). Learning and representation change. *Proc. AAAI '87.* (pp. 511-515). Cambridge, MA: Morgan Kaufmann.

Smith, E. & Medin, D. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Tcheng, D., Lambert, B., & Lu, S. C-Y. (1989). Generalized recursive splitting algorithms for learning hybrid concepts. *Proc. Sixth International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Utgoff, P. (1986). *Machine learning of inductive bias.* Dordrecht: Kluwer.

Utgoff, P. (1988). Perceptron trees: a case study in hybrid concept representation. *Proc. AAAI '88.* (pp. 601-606). San Mateo, CA: Morgan Kaufmann.