MAINTAINING THE UTILITY OF LEARNED KNOWLEDGE USING MODEL-BASED ADAPTIVE CONTROL

BY

LAWRENCE BRUCE HOLDER, JR.

B.S., University of Illinois, 1986
M.S., University of Illinois, 1988

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois

MAINTAINING THE UTILITY OF LEARNED KNOWLEDGE USING MODEL-BASED ADAPTIVE CONTROL

Lawrence Bruce Holder, Jr., Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1993
Larry A. Rendell, Advisor

The overfit problem in empirical learning and the utility problem in analytical learning both describe a common behavior of machine learning methods: the eventual degradation of performance due to increasing amounts of learned knowledge. Plotting the performance of the changing knowledge during execution of a machine learning method (the performance response) reveals similar curves for several methods. The performance response generally indicates a single peak performance greater than that attained by popular pruning techniques. The similarity in performance responses suggests a parameterized model relating performance to the amount of learned knowledge. Given this model, a model-based adaptive control (MBAC) approach can be used to update the model based on feedback from the performance element and make control decisions regarding the amount of knowledge to be learned or unlearned.

In view of the large number of alternative learning methods, a more general utility problem exists in determining not only the correct amount of learned knowledge, but also the correct method for learning this knowledge. Relying too heavily on one particular learning method may result in less than optimal performance achievement. Overcoming this *general utility problem* requires a new control mechanism for determining the correct learning method and amount of learned knowledge in order to achieve the performance objectives of the task. Maintaining models for several learning methods allows the MBAC approach to decide the appropriate type of learning, in addition to the amount.

Experimentation analyzes the ability of the MBAC approach to converge upon the peak of the performance response and avoid generation of low utility knowledge. Results indicate that a quadratic model is sufficient to fit the peak of the performance response and that MBAC using the quadratic model performs well at selecting the best learning method for a given learning task. More formal analysis of the performance response supports the quadratic model for controlling how much knowledge to learn as opposed to which knowledge.

DEDICATION

To my parents – Jeanette and Lawrence, Sr. – for the freedom to dream and the discipline to achieve.

ACKNOWLEDGEMENTS

Many people deserve acknowledgement for their contributions to my research. Foremost, I would like to thank my advisor, Larry Rendell, for providing indispensable feedback and guidance during my doctoral research. I would also like to thank the remaining members of my committee, Jerry DeJong, Mehdi Harandi, Arthur Baskin and Jean Ponce, for their helpful suggestions. The combined guidance of the committee members has greatly improved the impact of my research. Thanks also to Robert Stepp for guidance during the preliminary stages of this research.

Many of my colleagues also deserve credit for their help and support. Members of the Inductive Learning Group at the University of Illinois provided many useful comments during group meetings and individual discussions. They are Gunnar Blix, Eduardo Perez, Harish Ragavan, Jay Scott, Raj Seshu, Larry Watanabe, and Der-Shung Yang. Special thanks is due David Tcheng for reviewing the dissertation and providing several insightful comments. I would also like to thank Brad Whitehall and Bharat Rao for their comments, and my officemates, Ken Smith and Ken Tarbell, for their friendly support.

Perhaps the most essential help to the completion of this dissertation was provided by Diane Cook. Her contributions and reviews of my research are only equaled by her love and support during my last four years in graduate school.

I would especially like to thank my family for their enthusiastic and loving support throughout all my academic endeavors. And I would like to thank Sharon Collins for her ever-pleasant assistance.

Finally, I would like to acknowledge Ray Mooney, Jude Shavlik and Carl Kadie for their implementations of several machine learning programs used in this research.

This research was partially supported by the National Science Foundation under grant NSF IRI 8822031.

TABLE OF CONTENTS

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem

One of the main goals of machine learning research is the development of methods for generating knowledge that improves performance on some task. For example, empirical learning methods typically use a set of training examples to generate knowledge for improving classification accuracy on unseen examples. Analytical (explanation-based) learning methods use a single example to generate knowledge for improving the problem-solving speed on unseen examples. Machine learning research has developed several empirical and analytical learning methods that demonstrate performance improvements due to learned knowledge.

Unfortunately, more in-depth experimentation with these methods reveals that the performance improvement is not monotonic. As the methods generate more and more knowledge, the performance for which they were designed to improve, eventually degrades. In empirical learning, this phenomenon relates to overfit. As empirical methods generate more knowledge, they may increase the complexity of the hypothesis. For example, some empirical learning methods adapt a parameterized model to the training data. As the number of parameters in the model becomes a sizable fraction of the number of data, the method fits the parameters according to trends in the training examples that do not occur in unseen examples. In analytical learning, this phenomenon is known as the utility problem. As analytical learning methods generate more knowledge, they increase the amount of time needed to consider the application of the knowledge. The method eventually learns low-utility knowledge whose retention cost outweighs the performance benefits. In both learning paradigms, the degradation of knowledge

utility results from generating knowledge that does not contribute to performance improvement for the given task.

In order to avoid the knowledge utility problem, the learning method must determine the correct subset of the learnable knowledge that maximizes performance. Of course, trying all possible subsets is computationally infeasible. Therefore, most learning methods generate knowledge from specific to general or general to specific. Given that the learning method acquires knowledge in order of generality (specificity), avoiding the utility problem reduces to generating the correct *amount* of learned knowledge. This thesis addresses the problem of controlling a learning method in order to generate the correct amount of learned knowledge that improves performance without degradation. The more refined problem of determining *which* knowledge to retain is beyond the scope of this thesis, although Chapter 6 discusses the issue as a future direction for this research.

In view of the large number of alternative methods available for improving a given performance dimension (e.g., classification accuracy or problem-solving speed), a more general utility problem exists in determining not only the correct amount of learned knowledge, but also the correct method for learning this knowledge. For example, one learning method may achieve better performance than another method for a particular task, yet the reverse may be true for a different task. A similar situation exists for different settings of external parameters of a particular method. Relying too heavily on one particular learning method may result in less than optimal performance achievement. Overcoming this *general utility problem* requires a new control mechanism for determining the correct learning method and amount of learned knowledge in order to achieve the performance objectives of the task.

1.2 Proposed Solution

As Chapter 2 will demonstrate, a common behavior exists among several machine learning paradigms. Figure 1.1 illustrates this behavior, which is a result of the general utility problem inherent in learning methods that attempt to optimize some dimension of performance on unseen examples of a task. The performance initially increases, but then eventually degrades. Because this behavior is common among different learning paradigms, a control mechanism can use a model of this behavior to determine the amount of learned knowledge necessary to achieve

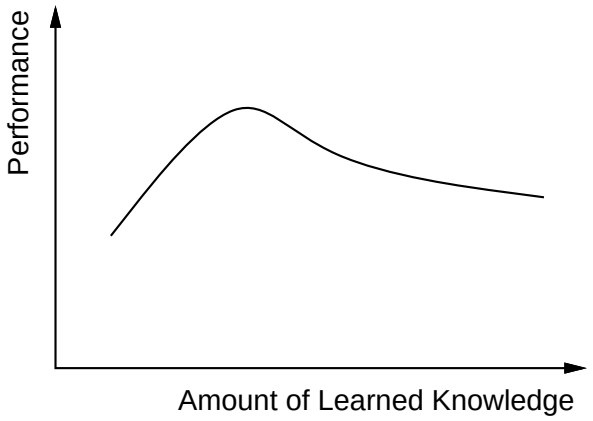

Figure 1.1: Relationship between performance and amount of learned knowledge for a learning method that suffers from the general utility problem.

a desired level of performance. Furthermore, the control mechanism can select appropriate learning methods according to the shape and certainty of their associated models.

This thesis investigates a model-based adaptive control (MBAC) approach to the general utility problem. Figure 1.2 illustrates the flow of control in the MBAC approach. The control mechanism utilizes performance feedback information from the performance element to decide the correct learning method and amount of learning for transforming existing knowledge in order to achieve the performance objectives of a task. MBAC is adaptive, because the decisions may change over time as the system acquires more experience in a particular task. The adaptability of MBAC comes from the maintenance of multiple models that relate performance to the amount of learned knowledge – one model for each combination of tasks, performance dimensions and learning methods. Each model is a parameterized curve that fits the behavior of Figure 1.1.

As an example of the MBAC approach, suppose the task is to determine the class (positive or negative) of an example. The performance objective is to maintain accuracy at 95%. The initial knowledge consists of a set of training instances. The available knowledge transformations are a decision tree induction method (ID) that transforms knowledge by extending the decision tree and a neural network method (NN) that transforms knowledge by performing another n cycles on the network. Therefore, MBAC defines two models: one for classification accuracy on this task as a function of decision tree size for ID, and one for classification accuracy on this task as a function of the number of cycles for NN.

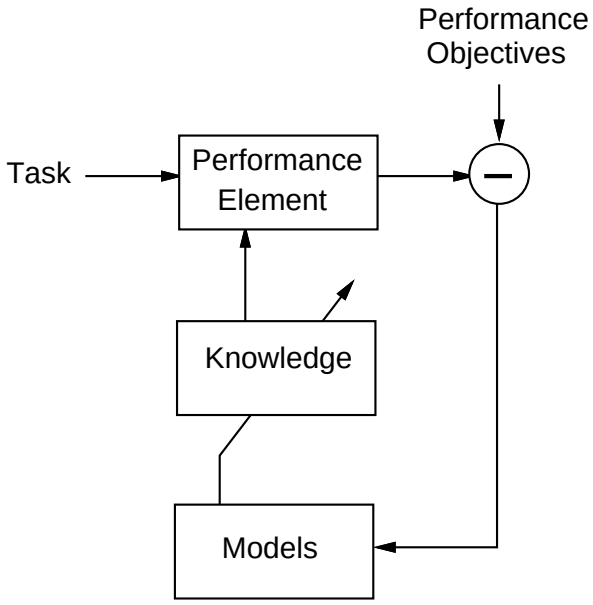

Figure 1.2: Model-Based Adaptive Control

At first, MBAC selects a transformation at random (e.g., one extension to the decision tree or n cycles of the network), because no data exists for fitting the parameterized model. After each transformation, the performance element uses the resulting knowledge to measure accuracy on a set of testing instances specified by the task. The measured accuracy combined with the most recent transformation provides a data point for fitting the model. Eventually, MBAC has a model for both ID and NN. Using these models, MBAC selects a transformation method and amount of transformation according to the model's certainty and ability to attain the desired performance objective. MBAC continues selecting transformations until either the measured performance satisfies the performance objectives or all models recommend no transformations.

1.3 Benefits

The general utility problem behavior of Figure 1.1 and the MBAC approach based on this behavior offer several benefits for individual learning methods and for the integration of several learning methods into a multi-strategy learning system. The common behavior in the relationship between performance and the amount of learned knowledge provides a new mechanism for controlling a learning method that suffers from the general utility problem. This mechanism recommends a decomposition of the learning method into simpler knowledge transformations

that enable perception of the relationship between performance and knowledge. Equipped with transformations making smaller changes in performance, the control mechanism can avoid the generation of low utility knowledge by performing transformations (learning and unlearning knowledge) in order to converge upon the peak of the curve in Figure 1.1. The same control mechanism is applicable to a variety of learning methods.

Using one parameterized model to describe the behavior of multiple learning methods simplifies the integration of these methods. Instead of integrating on the basis of a common knowledge representation, a multi-strategy learning system can integrate on the basis of the performance/knowledge relationship while maintaining individual knowledge representations for each method. Estimating the certainty of the models with respect to the model data provides a means of ordering the methods according to their likelihood of attaining performance objectives. Furthermore, by attaching a resource cost function to each transformation, MBAC can trade off attainable performance with resource expenditure in the case of limited resources.

The MBAC approach extracts the performance objectives implicit in the learning methods and explicitly defines them external to the methods. Explicit performance objectives simplify the decomposition of learning methods. This explicitness also allows multiple performance objectives for one task, changing performance objectives, and changing performance elements. Through the use of parameterized models, MBAC adapts the knowledge according to the changing performance environment.

1.4 Outline

Chapter 2 discusses the general utility problem and demonstrates the existence of the problem in several machine learning paradigms. The chapter begins by defining the performance response, a tool for analyzing the general utility problem in learning. Figure 1.1 is an example of a performance response. Next, the chapter considers three empirical learning paradigms: splitting methods, agglomerative methods and neural network methods. Performance response curves for these methods confirm the existence of the general utility problem. Chapter 2 also considers several analytical learning methods whose susceptibility to the general utility problem has been demonstrated by other researchers. An actual performance response of a particular analytical learning method further confirms the commonality of the behavior in Figure 1.1. The chapter

concludes with a discussion of the trends identified by the aforementioned performance responses and a more formal analysis of the general utility problem in several of the paradigms.

Chapter 3 describes the MBAC approach. First, the chapter defines the approach and outlines the adaptive control algorithm. The remainder of Chapter 3 discusses the issues involved in each aspect of the algorithm. The four main issues are the integration of the diverse knowledge representations used by machine learning methods, the properties of the operations that transform this knowledge, the expression of the performance objectives that drive the MBAC approach, and the properties of the models that form the foundation of MBAC.

Chapter 4 describes an implementation of the MBAC approach and presents experimental results. The first experiment demonstrates MBAC's ability to converge to the peak of the performance response and avoid the generation of low utility knowledge using a quadratic model of the performance response. The second experiment compares three different estimates of model certainty and indicates that the standard deviation of the model is superior. The third experiment shows the ability of MBAC to select an appropriate learning method according to the certainty of the associated models. Experiment 4 demonstrates MBAC's adaptive behavior while refining the models and converging to the peak of the performance response. Experiment 5 demonstrates the use of previously adapted models to make decisions about new tasks that have little or no model information. The chapter concludes with a summary of experimental results and overall evaluation of the MBAC method.

Chapter 5 describes work related to the MBAC approach. Related work includes research on approaches to utility control, multiple-method control and adaptive control. Chapter 6 describes directions for future work, and Chapter 7 concludes with a summary of the results and contributions of the research.

Chapter 2

General Utility Problem

A primary goal of machine learning research is the development of autonomous methods for acquiring knowledge in order to improve performance on some task. In a perfect world where tasks are described by finite numbers of uniformly-distributed, error-free instances, knowledge acquired by machine learning methods increases performance on the task. In other words, the acquired knowledge has *utility* with respect to performance on the task. However, the world is not perfect. Tasks may have an unknown or infinite number of instances with non-uniform distributions and noisy descriptions. Knowledge acquired from such instances may have lower utility with respect to task performance.

As machine learning methods acquire increasing amounts of knowledge based on imperfect instances, the proliferation of low-utility knowledge increases, and performance degrades. The *general utility problem* in machine learning refers to the degradation of performance due to increasing amounts of learned knowledge [Holder, 1990]. This term derives from the *utility problem* used by Minton [1988b] to describe this phenomenon in analytical learning, but generalizes to other machine learning paradigms.

Other researchers have observed the ubiquity of the utility problem in machine learning paradigms. Carlson *et al.* [1990] compare the utility problem in analytical learning to the problems of noise and overfit in empirical learning. Etzioni [1988] alludes to the general utility problem as he proposes a hypothesis filter for all learning methods. The filter approves learned hypotheses only if they have high utility with respect to user-defined performance objectives. As with most current approaches to the utility problem analyzed in this chapter, these

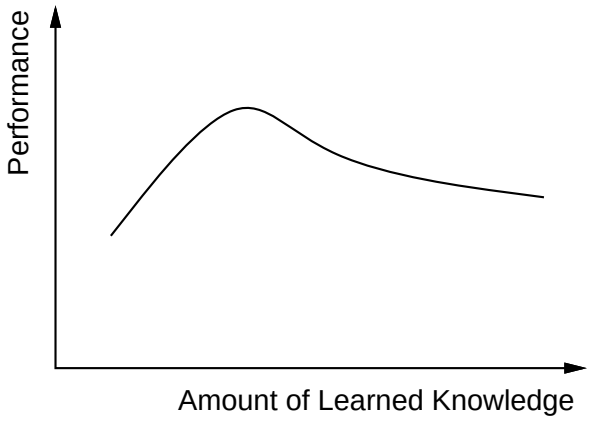

Figure 2.1: Performance response indicative of the general utility problem

two approaches depend upon a model of the relationship between performance and learned knowledge. The approaches require detailed knowledge of the performance environment. The proposed MBAC approach resides at the opposite end of the spectrum over performance environment knowledge. The performance response trend identified in this chapter provides a more general model of the relationship between performance and learned knowledge, and reduces MBAC's dependence on knowledge of the performance environment.

The first section of this chapter introduces the performance response, a tool for analyzing the general utility problem. Section 2.2 describes several empirical learning methods that suffer from the general utility problem and recent approaches for alleviating the problem. Section 2.3 covers the same areas for analytical learning methods. Section 2.4 analyzes trends uncovered in the previous sections and describes the role that these trends play in the proposed approach to the general utility problem. Section 2.5 provides a more formal analysis of the general utility problem in these methods.

2.1 Performance Response

A useful tool for analyzing the general utility problem in machine learning is the performance response. The *performance response* is the performance of the learned knowledge measured during the course of learning. Figure 2.1 illustrates the typical performance response of a learning method that suffers from the general utility problem.

The horizontal axis of the performance response measures the amount of learned knowledge. The units along this axis represent the change in learned knowledge made by a knowledge transformation. A knowledge transformation is a decomposition of the learning method into less complex operations affecting the learned knowledge. For example, one decomposition of a splitting algorithm is a single split, and one decomposition of a neural network learning algorithm is a cycle. Since a knowledge transformation may not always increase the amount of learned knowledge in terms of the size of the set of knowledge, an increase along this axis more generally represents a refinement of existing knowledge.

This approach does not attempt to formally define the amount of learned knowledge in terms of well-defined units (e.g., number of bits). Instead, the approach uses a measure that corresponds to a natural decomposition of the learning method and that implies the amount of actual learned knowledge. For example, Section 2.5.1 shows how the number of splits made by a decision tree induction method corresponds to the amount of knowledge represented by the decision tree. Experimentation in this chapter illustrates how increasing the amount of knowledge based on training data reduces the utility of this knowledge on unseen data. The choice of the measure of knowledge is arbitrary, but remains fixed in order to compare performance after each unit of learned knowledge. This method for selecting the measure of knowledge prevents a comparison between learning methods using different measures. Although the measures are incompatible, the relationship between performance and the measure of the amount of learned knowledge is similar for different learning methods. This chapter reveals the similarity, which forms the foundation of the MBAC approach for selecting appropriate learning methods and avoiding low-utility knowledge.

The vertical axis of the performance response measures the performance of the learned knowledge after each transformation. The measure of performance depends upon the learning method. Different methods attempt to improve different dimensions of performance. For example, a neural network primarily attempts to improve classification accuracy, while an explanation-based learning method attempts to improve problem-solving speed. Other performance measures on the learned knowledge include the complexity and storage cost of the knowledge. The classification accuracy of empirical learners and the problem-solving speed of analytical learners are the focus of this work.

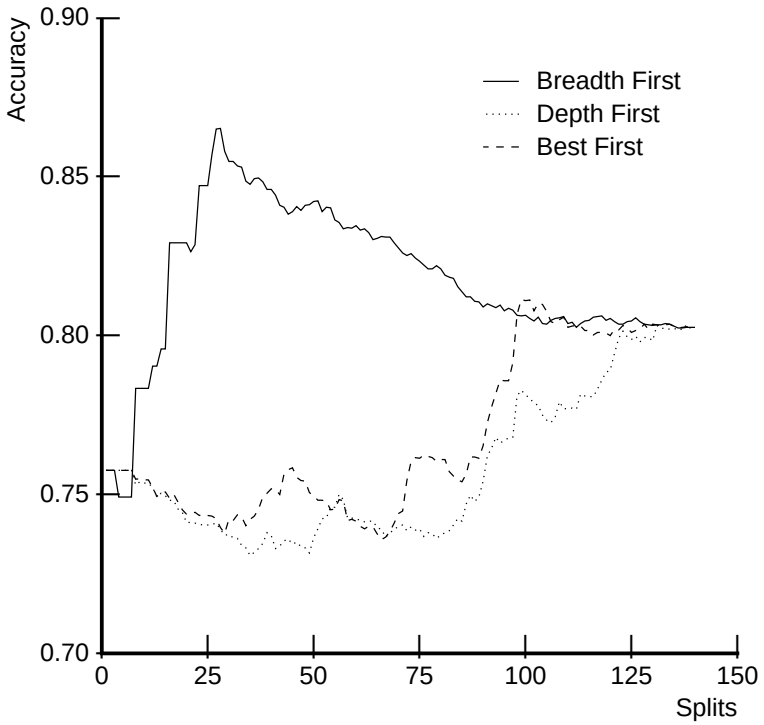

Figure 2.2: Performance responses for three traversals of a decision tree induced from the DNF2 domain.

For empirical learning the performance response curve plots the classification accuracy of the knowledge after each transformation. The knowledge transformations use a set of training data to transform the knowledge. The classification accuracy for the current knowledge is the quotient of the number of correctly classified instances in a separate testing set over the total number of instances in the testing set.

For analytical learning the performance response curve plots the inverse of the CPU time needed by the knowledge to solve a set of test problems. The knowledge transformations use a problem from a set of training problems to learn a new rule for improving the speed of the problem solver. The performance of the new knowledge (set of rules) is the inverse of the CPU time necessary for the new knowledge to solve the set of test problems.

As an example, Figure 2.2 illustrates three performance responses obtained from the ID3 empirical learner[1] on the DNF2 domain[2]. ID3 constructs a decision tree from the training data

[1]See Section 2.2.1.1 for an explanation of the ID3 program and additional performance response curves.
[2]See Appendix A for a description of the domains.

by splitting the data at a node. Splitting continues until satisfaction of a stopping criterion. Each performance response in Figure 2.2 represents a different traversal (node split order) of the decision tree. Performance is classification accuracy, and the amount of learned knowledge increases with the number of splits. Each performance response is an average over ten trials. Each trial consists of selecting random training and testing sets, generating the decision tree using the training set, and measuring accuracy after each split using the testing set. Unless stated otherwise, all performance responses shown in this thesis represent the average over ten trials.

As Figure 2.2 reveals, the order of the knowledge transformations is important for perceiving the desired performance response trend in Figure 2.1. Section 3.4 discusses this and other issues pertaining to decomposing learning programs into knowledge transformations. Before discussing the issues in Chapter 3, the remainder of this chapter presents performance response curves for both empirical and analytical learning paradigms. The results confirm that the performance response curves of many learning paradigms follow the trend indicative of the general utility problem.

2.2 Empirical Learning

Empirical learning attempts to induce general knowledge from a set of training data. The set of training data consists of classified examples (instances) of the desired concept. For this discussion the instances are assumed to be a set of propositional rules of the form:

$$([\text{feature} = \text{value}] \wedge [\text{feature} = \text{value}] \wedge \cdots) \rightarrow \text{class}$$

The induced general knowledge may also be of this form. In the case of splitting algorithms, the general knowledge is often in the form of a decision tree. For neural network learners, the general knowledge is in the form of a network of nodes and weighted links. Typically, the performance environment uses a separate set of test data to evaluate the general knowledge produced by the empirical learner.

The general utility problem in empirical learning relates to the overfit problem. Overfit occurs when the learning method identifies errant patterns in the training data. Errant patterns may arise due to noise in the training data or inadequate stopping criteria of the method. As

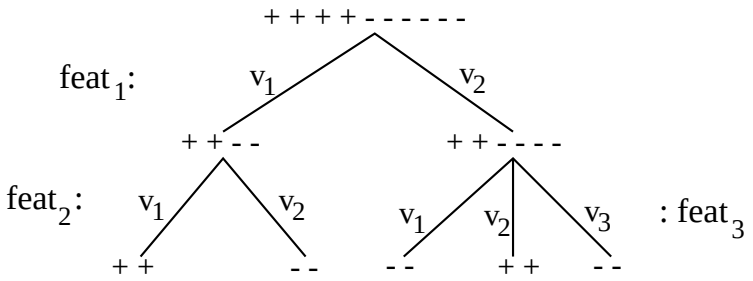

Figure 2.3: ID3 decision tree.

demonstrated below, splitting, agglomerative and neural network learning methods suffer from overfit.

2.2.1 Splitting Methods

Splitting methods recursively split the set of training data by choosing an appropriate feature or feature-value pair. The main parameters of the method are the splitting criterion and the stopping criterion. Overfit results from an inappropriate stopping criterion which allows the method to perform too much splitting. The following sections describe two splitting methods (ID3 and PLS1) and illustrate their tendency to overfit.

The knowledge produced by a splitting method can be represented as a decision tree. An example of a decision tree produced by the ID3 method (see next section) is shown in Figure 2.3. The learned knowledge changes every time the method makes a split; therefore, one choice for the x-axis of the performance response is the number of splits. The y-axis (performance) measures the classification accuracy of the knowledge after each split, as measured using a separate set of test data. The axes of the performance responses for the two splitting methods discussed below (ID3 and PLS1) follow this arrangement. For the empirical learning methods discussed in this chapter, the y-axis will always measure classification accuracy.

2.2.1.1 ID3

The ID3 (Induction of Decision trees) program developed by Quinlan [1986] induces decision trees by recursively splitting the given set of training instances. Although ID3 has many variants, this discussion assumes the training instances fall into one of two classes: positive (+) and negative (−). Figure 2.3 shows the type of decision tree built by ID3. At the root of

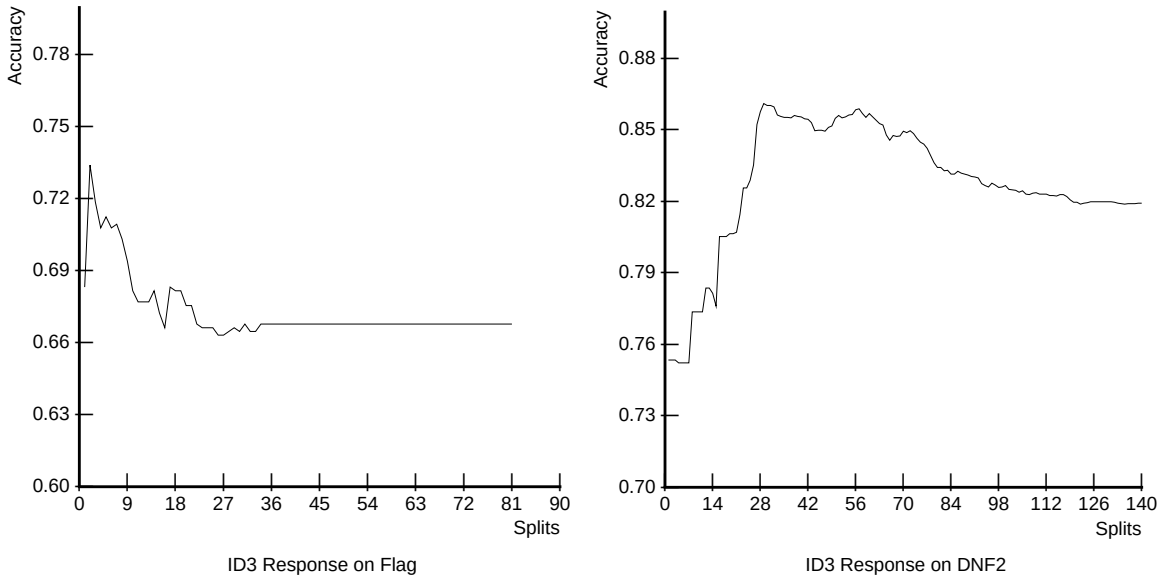

Figure 2.4: Performance response of ID3.

the tree, ID3 selects a feature to split the training instances according to the different values of the feature. Splitting continues until all nodes at the frontier of the tree are "pure" nodes, i.e., all instances at the node are in the same class. ID3 splits an impure node by selecting a feature, creating child nodes for each value of the feature, and splitting the instances into the child nodes according to their value for the split feature. The split feature at a node is a feature not yet used as a split along the path to the node and minimizing the mutual information MI,

$$MI = -\sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \left(\frac{p_i}{p_i + n_i} log_2 \frac{p_i}{p_i + n_i} + \frac{n_i}{p_i + n_i} log_2 \frac{n_i}{p_i + n_i} \right)$$

where i ranges over the values of the feature, p and n are the number of positive and negative instances at the node, and p_i and n_i are the number of positive and negative instances at the node having value i for the feature.

Although the choice of splitting criterion has little effect on the behavior of ID3 [Mingers, 1989b], the choice of stopping criterion greatly affects the performance of the final decision tree [Mingers, 1989a]. The ID3 performance response in Figure 2.4 plots the accuracy of the decision tree on a separate set of testing instances after each split. Splits are performed in a breadth-first order, deferring overfit to the later splits. Figure 2.4 shows the performance response of ID3 on the Flag and DNF2 domains[3] using the node-purity stopping criterion above. As the

[3]See Appendix A for descriptions of the domains.

figure illustrates, this stopping criterion causes overfit, and the performance responses follow the trend of Figure 2.1.

Two tree-pruning techniques have been developed to combat overfit: pre-pruning and post-pruning. Pre-pruning constrains the stopping criterion to prevent splitting of impure nodes when no feature provides a significant increase in information resulting from a split. Post-pruning uses the pure-nodes stopping criterion to generate the decision tree, but then removes subtrees of the resulting tree to improve performance.

Quinlan [1986] developed a pre-pruning technique for ID3 based on the chi-square statistic:

$$\chi^2 = \sum_{i=1}^{v} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

where

$$p'_i = p \cdot \frac{p_i + n_i}{p + n} \quad \text{and} \quad n'_i = n \cdot \frac{p_i + n_i}{p + n}$$

The p'_i and n'_i represent estimates of the number of positive and negative instances having value i for the feature if the feature is irrelevant to the class value, where i ranges from one to the number of values v of the feature. The chi-square statistic allows consideration of the hypothesis that the feature value is independent of the class. The value of χ^2 and the number of degrees of freedom $(v-1)$ are used to estimate the probability with which one can reject this hypothesis [Freund, 1988]. If the hypothesis cannot be rejected with very high confidence, say 99%, then the feature will not be considered for splitting.

Figure 2.5 shows the performance response of ID3 with chi-square pre-pruning on the Flag and DNF2 domains. Responses are plotted for confidence values of 99% and 99.9%. Although chi-square pre-pruning reduces the number of splits, overfit behavior is still evident. Increasing the confidence value may further reduce the number of splits, but does not eliminate overfit.

Breiman *et al.* [1984] state that most pre-pruning techniques are unsatisfactory due to their dependence on a user-supplied parameter (e.g., the 99% used for chi-square pre-pruning). As an alternative, they propose a post-pruning technique that splits the decision tree to pure nodes and then prunes back. Mingers [1989a] compares several post-pruning techniques and concludes that Quinlan's [1987] reduced-error post-pruning is among the best. With reduced-error post-pruning, subtrees are removed from the original decision tree until accuracy decreases on a separate set of pruning instances.

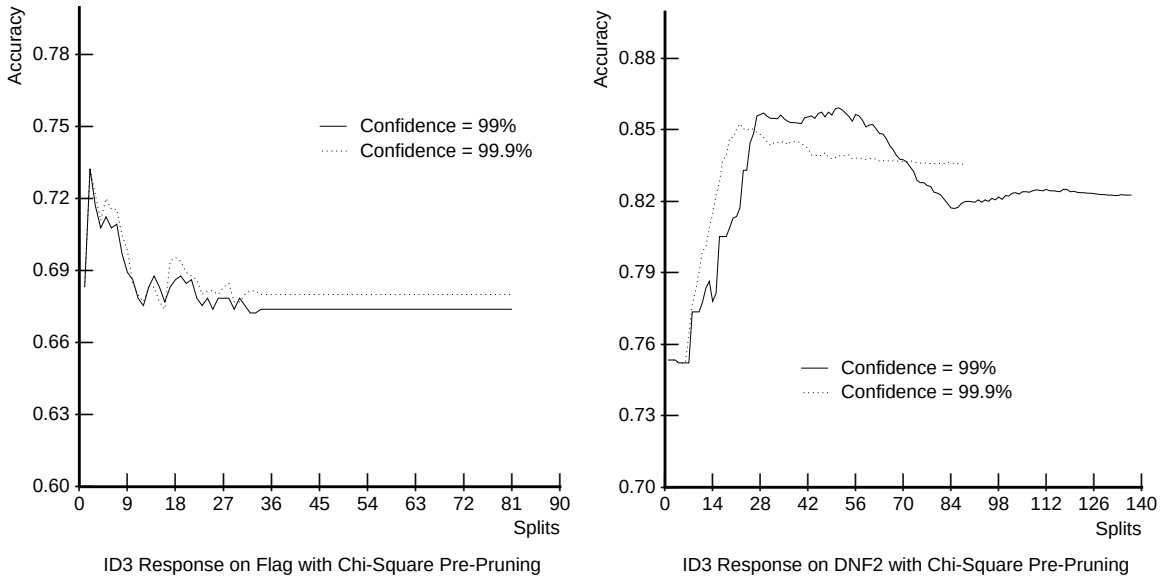

Figure 2.5: Performance response of ID3 with chi-square pre-pruning.

Figure 2.6 shows the performance response of ID3 with reduced-error pruning on the same domains. The dotted vertical line marks the point at which the full tree has been generated, and pruning begins. The reduced-error pruning alleviates most of the overfit, but on average the accuracy of the resulting tree is less than the peak accuracy of the performance response (see Table 2.1 in Section 2.4).

2.2.1.2 PLS1

The PLS1 program (Probabilistic Learning System) developed by Rendell [1983] is similar to ID3 in that the initial hypothesis is the most general, and the method specializes the hypothesis according to the training data. PLS1 specializes the hypothesis by recursively dividing the instance space into hyper-rectangular regions. This is equivalent to performing a binary split on a particular feature-value pair. Figure 2.7 shows a sample hyper-rectangle and the equivalent decision tree. Instances with values less than or equal to the split value for the feature follow one branch, and the remaining instances follow the other branch. A node in the resulting decision tree represents a region of instance space constrained by the path leading to the node. PLS1 splits a region by selecting a hyperplane that divides one dimension (feature) of the instance space within the region. PLS1 chooses the hyperplane that maximizes a probabilistic dissimilarity measure d:

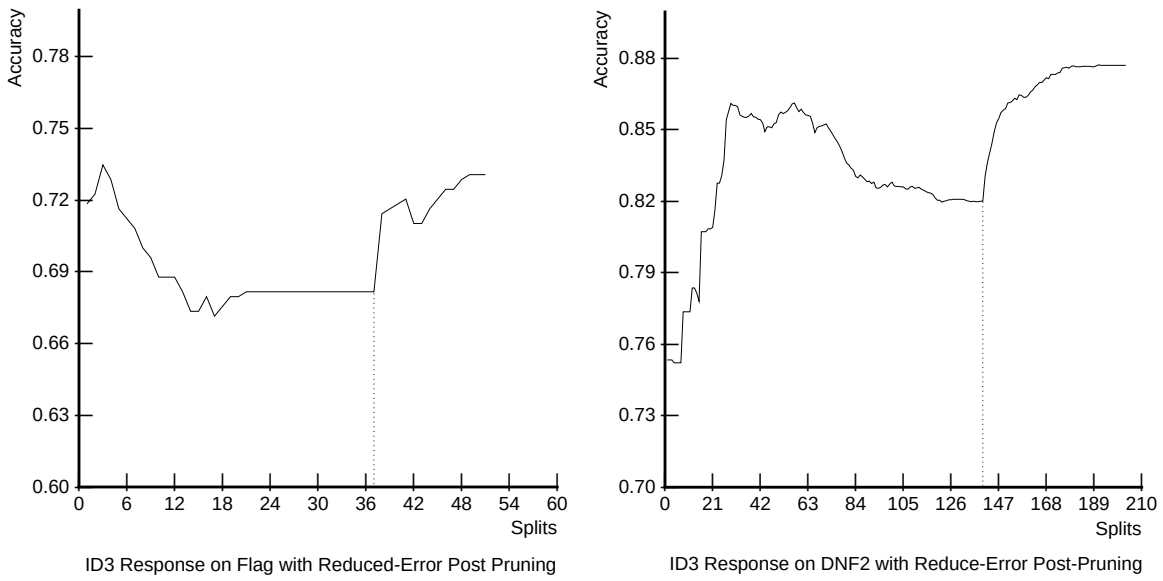

Figure 2.6: Performance response of ID3 with reduced-error post-pruning.

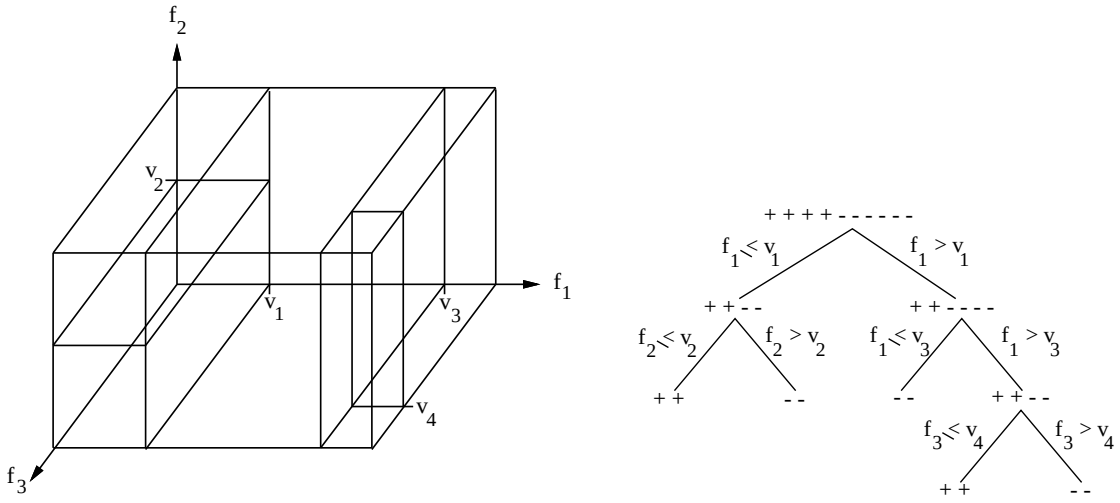

Figure 2.7: PLS1 hyper-rectangles and corresponding decision tree.

$$d = \mid log_2 u_1 - log_2 u_2 \mid - t_\alpha \times log_2(e_1 e_2)$$

where

$$u_1 = \frac{p_1}{p_1 + n_1},\ u_2 = \frac{p_2}{p_2 + n_2},\ e_1 = 1 + \frac{1}{\sqrt{p_1 + n_1}},\ e_2 = 1 + \frac{1}{\sqrt{p_2 + n_2}}.$$

The p_1, n_1, p_2 and n_2 are the number of positive and negative instances in the two regions resulting from the split by the hyperplane under evaluation. The t_α constant represents the number of standard deviations, i.e., the desired degree of confidence. Typical values for t_α are between 1 and 2.

Since PLS1 chooses to split only if the maximum dissimilarity is positive, the t constant can be used to restrict the amount of splitting. Thus, t_α in PLS1 plays a role analogous to the confidence level in the chi-square pre-pruning technique for ID3. Figure 2.8 shows the performance responses of PLS1 on the same domains used for ID3. Each plot displays a response curve for three different values of t_α: 1.0, 1.5 and 2.0. As with the chi-square pruning of ID3, increasing t reduces the number of splits made by PLS1, but the tendency to overfit is still evident.

2.2.2 Agglomerative Methods

An agglomerative (or set-covering) method for empirical learning constructs a hypothesis which describes a subset of the training instances, and then applies the same method on the remaining training instances. Alternative hypotheses are evaluated by user-supplied criteria or statistical measures. An agglomerative method differs from a splitting method in that the splitting method uses a divide-and-conquer approach; whereas, the agglomerative method uses a separate-and-conquer approach. Another difference between the two methods is that splitting methods specialize from a hypothesis covering all examples; whereas, agglomerative methods generalize from a hypothesis covering no examples.

Since agglomerative methods typically learn disjunctive normal form (DNF) expressions for the hypotheses, the amount of learned knowledge varies over two dimensions: the number of literals per disjunct and the number of disjuncts. As the number of literals per disjunct increases, the disjunct describes a smaller region of the instance space, and more disjuncts are

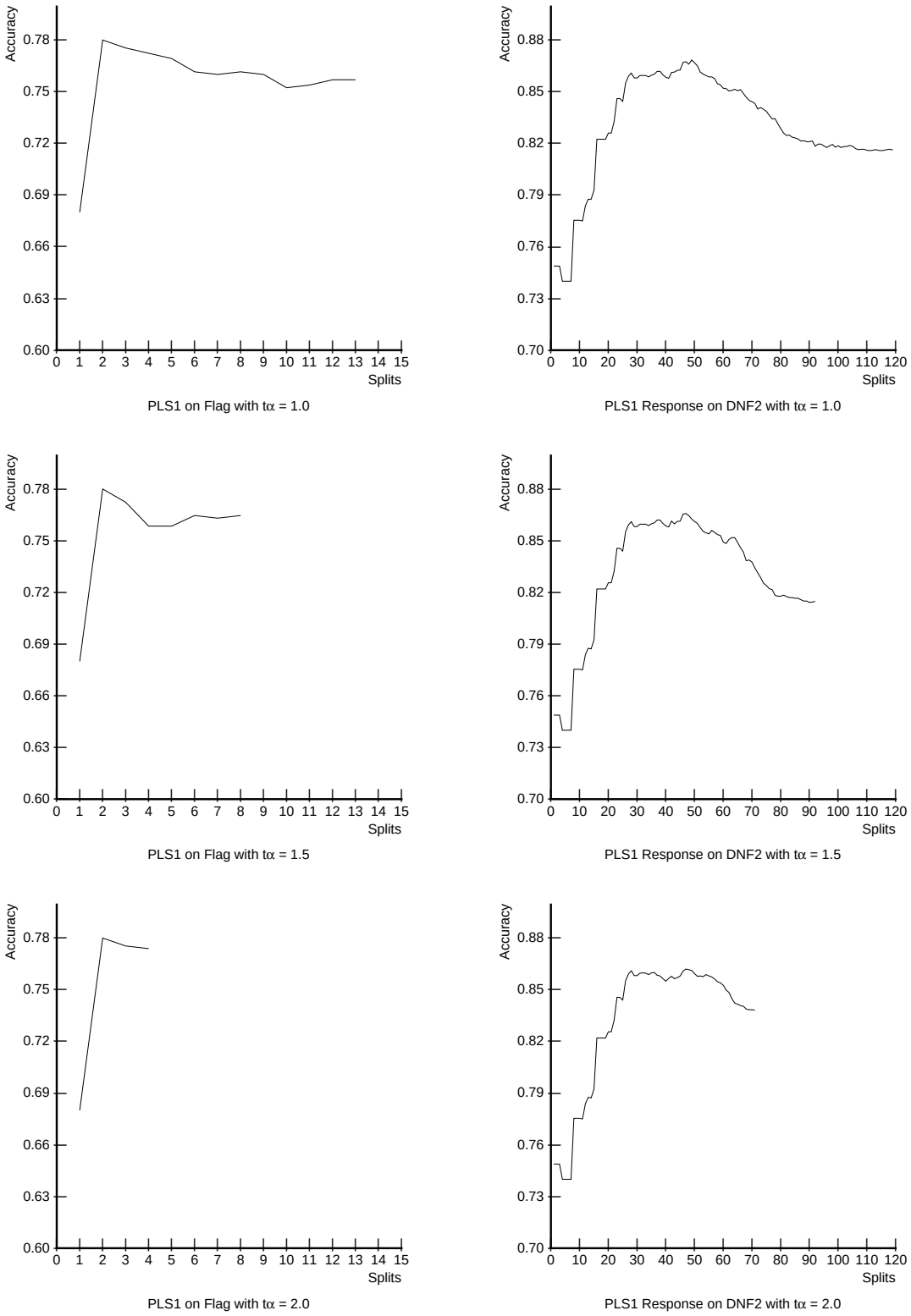

Figure 2.8: PLS1 performance response.

necessary to describe the portion of the instance space represented by the training instances. Eventually, each disjunct will have a large number of literals and will describe a small number of instances. In this extreme, overfit occurs due to noise in the training instances and a strong bias preferring overly specific hypotheses. Since an increase in one dimension of the amount of learned knowledge (number of literals) implies an increase in the other dimension (number of disjuncts), only one dimension need be monitored for the performance response.

The following sections discuss experiments performed by other researchers on agglomerative methods. The experiments indicate the presence of the general utility problem in these methods. The dimension used to measure the amount of learned knowledge is the number of disjuncts in the induced hypothesis.

2.2.2.1 AQ

During experimentation with the AQ system (specifically, AQ15 [Michalski *et al.*, 1986]), Michalski found that repetitive application of AQ can yield less accurate hypotheses than a more conservative application strategy combined with a more flexible inference mechanism than exact matching [Michalski, 1989]. The AQ method finds a conjunctive description that covers as many positive examples as possible without covering any negative examples. Positive examples not covered by previously-generated descriptions are used as input for another execution of AQ. This procedure continues until the descriptions (disjuncts) generated by AQ form a hypothesis in disjunctive normal form (DNF) that covers all the positive examples and none of the negative examples.

Michalski compared the accuracy of the *complete* DNF hypothesis produced by AQ to truncated versions of the same hypothesis. The first truncated version of the hypothesis consists of the single disjunct covering the most examples (*best disjunct*). The second truncated version of the hypothesis consists of only those disjuncts covering more than one unique example (*unique* > 1). The truncated hypotheses use a simple matching procedure for classifying uncovered and multiply-covered examples (see [Michalski, 1989] for details).

Although based on only four points, Figure 2.9 approximates the performance response of AQ in three medical domains (Lymphography, Breast Cancer and Primary Tumor) averaged over four trials. The DNF hypotheses are of the form

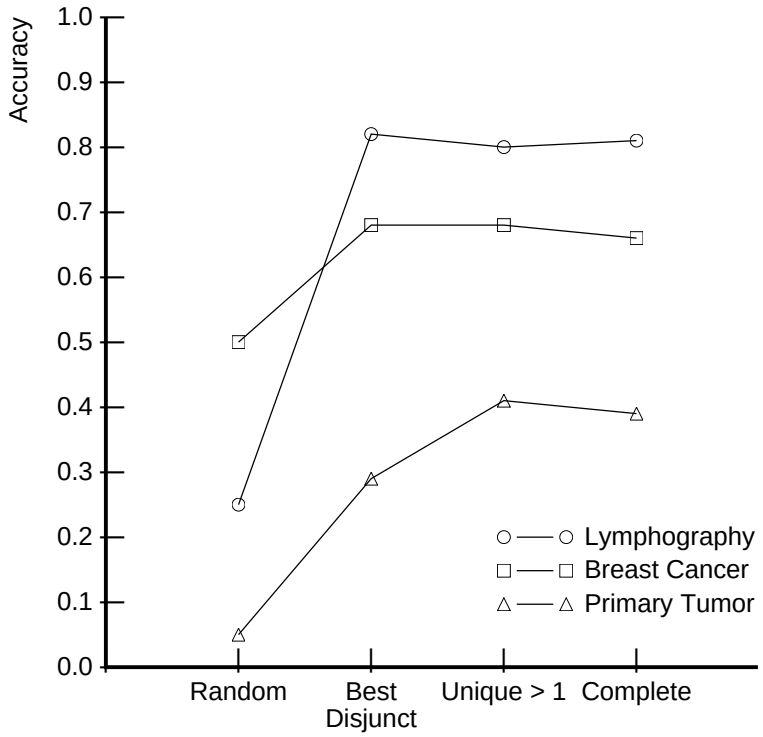

Figure 2.9: Performance response of AQ in three medical domains.

$$(\text{disjunct}_1 \vee \cdots \vee \text{disjunct}_n) \rightarrow \text{class}$$

The number of disjuncts in the DNF hypothesis increases along the increasing x-axis. The *random* point represents zero disjuncts on the left of the rule, and a randomly-selected class on the right. The accuracies for the random points are 0.5, 0.25 and 0.05 for Breast Cancer, Lymphography and Primary Tumor (respectively), because the respective number of classes are 2, 4, and 20. The *best disjunct* point represents the one best disjunct covering the most positive training instances. The *unique > 1* point represents more than one disjunct, but less than the number of disjuncts in the *complete* DNF hypothesis. Figure 2.9 demonstrates that AQ also suffers from the general utility problem with increasing numbers of disjuncts, and the response curves indicate the same trend as in Figure 2.1.

2.2.2.2 CN2 and Small Disjuncts

The CN2 induction program developed by Clark and Niblett [1989] is another agglomerative empirical learning method. Instead of a disjunctive normal form hypothesis, CN2 produces a decision list: an ordered list of rules in the form *complex* $\rightarrow$ *class*. A *complex* is a conjunction of selectors of the form *[feature = value]*. CN2 proceeds by finding the best complex for distinguishing classes in the set of training instances according to two information-theoretic measures. CN2 adds the best complex to the end of the decision list, removes from the training data those instances covered by the complex, and begins another search for the best complex using the remaining training instances. The class implied by each complex is the majority class of the training instances covered by the complex.

Analyzing the hypotheses produced by CN2, Holte *et al.* [1989] reveal that the accuracy of the hypothesis degrades with the addition of small disjuncts. Small disjuncts are complexes covering a small number of examples. Because they are motivated from a small number of examples, small disjuncts are typically more error prone than large disjuncts. Therefore, CN2 suffers from the general utility problem due to the increasing amounts of low utility (small disjunct) knowledge.

Holte *et al.* consider three approaches to the problem. One approach eliminates all small disjuncts. However, this approach may eliminate significant small disjuncts that cover unusual

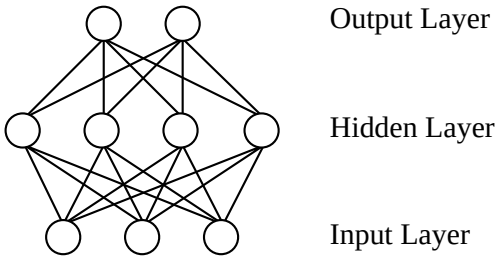

Figure 2.10: Three layer network

examples. Also, the error of the resulting hypothesis may be worse due to misclassification of these examples by other disjuncts or default rules. A second approach eliminates only undesirable disjuncts. Significance testing and error estimation offer measures of the desirability of a disjunct. The third approach uses a different bias for small and large disjuncts. Large disjuncts use a maximum generality bias, whereas small disjuncts use a selective specificity bias that specializes the disjunct so that it covers only 25% of the examples from other classes. Holte *et al.* present empirical evidence indicating the superiority of the different-bias approach used within the CN2 program.

2.2.3 Neural Network Methods

A neural network consists of two or more layers of interconnected units. Weights reside on each connection, and a unit produces a signal that is a function of the weighted input signals to the unit. Networks contain an input layer whose signals are derived from the feature values of examples, and an output layer that produces a prediction of the class of the example. Neural networks learn from training data by presenting the feature values of an instance to the input layer, comparing the output layer's prediction to the instance's class, and updating the connection weights according to the difference.

Network layers other than the input and output layers are called hidden layers. Figure 2.10 shows a network with one hidden layer containing four hidden units. One method for updating the weights in such a network is *error back-propagation* [Rumelhart *et al.*, 1986]. This method first computes the error between the output signal generated by an instance and the known class of the instance. Then, the error propagates back through the network, modifying the weights so as to alleviate the error. The change in the weight on the connection from unit i to unit j can be expressed as

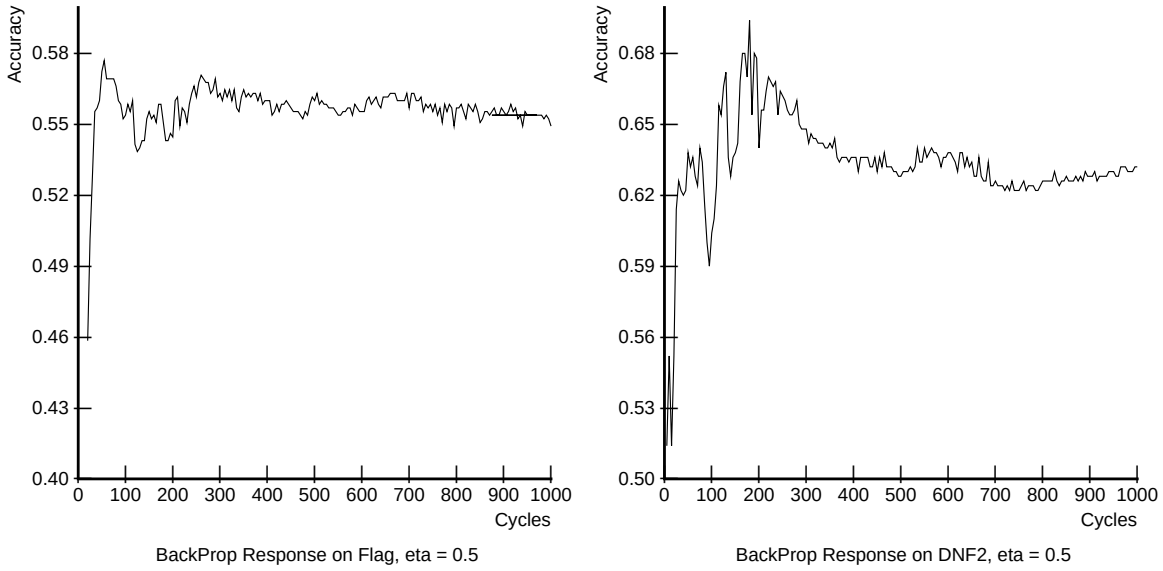

Figure 2.11: BackProp performance response.

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi},$$

where p represents the particular training instance pattern at the input units, o_{pi} is the output signal of unit i for pattern p, δ_{pj} is the error signal from unit j for pattern p, and η is the learning rate. Each error back-propagation pass through the set of training instances is called a *cycle*.

As the number of cycles increases, the network more accurately classifies the training instances. However, overfit eventually occurs as the network learns the training instances too precisely, degrading accuracy on a separate set of testing data. To analyze the overfit of the back-propagation neural network, the performance response measures accuracy of the network after every five cycles.

Figure 2.11 shows the performance response of the error back-propagation neural network (BackProp) on the Flag and DNF2 domains. For these experiments, the learning rate η was set at 0.5, and the network contained one hidden layer with four units. The choice of learning rate was arbitrary, because with a high enough learning rate, the number of hidden units determines the complexity of the function learnable by the network and, therefore, the extent of possible overfit [Karnin, 1990]. The output layer has two units, one for each of the two classes. The input

layer has one unit for each feature-value pair in the domain. The network correctly classifies an instance if the output signals are within 0.1 of their desired values.

The BackProp response on the Flag and DNF2 domains follows the general utility problem trend as in Figure 2.1. Table 2.1 in Section 2.4 reveals that on average the network at the initial peak performs better than the final network.

2.3 Analytical Learning

Research on analytical (explanation-based) learning techniques began to focus more attention on performance with the appearance of Keller's work on the definition of operationality [Keller, 1988]. Analytical techniques learn from a single example by proving the example is an instance of the concept to be learned. The proof terminates when the leaves of the proof tree are all operational predicates. The proof tree is then generalized, yielding an operational description of the concept. Earlier work on explanation-based learning defined an operational concept as one whose description is composed from a set of predicates deemed easy to evaluate [Mitchell *et al.*, 1986; DeJong and Mooney, 1986]. Keller points out that operationality is more intimately related to the performance element and the desired performance improvement. The increased attention on performance has led to the reevaluation of several analytical learning systems and the observation that performance may degrade with repeated application. Because explanation-based learning methods acquire correct knowledge, increased performance corresponds to faster problem solving.

The following sections describe three analytical learning systems, their susceptibility to the utility problem, and approaches to alleviating the problem. Section 2.3.4 illustrates the performance response of a simple analytical learner in two planning domains.

2.3.1 PRODIGY

In experimentation with the MORRIS analytical learning system, Minton found that performance eventually degrades with increasing numbers of learned macro-operators [Minton, 1985]. After solving a problem, the system creates a new operator capable of effecting the solution path in one step. However, as the number of macro-operators increases, the cost of determining the applicability of an operator may outweigh the benefits of applying, and thus, retaining the

operator. This phenomenon eventually degrades the problem-solving speed performance that MORRIS was designed to improve.

Minton called this phenomenon the *utility problem* and offered the PRODIGY system as a solution [Minton, 1988a]. PRODIGY learns control rules to improve problem-solving performance based on explanations of success and failure in actual problem-solving solution traces. The system maintains empirical estimates of match costs, application savings and frequency of application for each rule. These estimates are used to compute a utility value for the rule:

Utility = (AvrSavings x ApplicFreq) - AvrMatchCost

where
AvrMatchCost = average time cost of matching rule
AvrSavings = average time savings when rule is applicable
ApplicFreq = probability that rule is applicable when tested

If a rule's utility value becomes negative, PRODIGY discards the rule. Minton found that maintenance of a rule's utility value and compression of the rule's conditions result in a substantial performance improvement. Performance response data was unavailable for PRODIGY, but Section 2.3.4 shows performance responses for a simple forward-chaining problem solver while learning macro-operators.

2.3.2 SOAR

Experimentation on the SOAR system has uncovered similar results [Tambe and Newell, 1988]. SOAR compiles problem-solving episodes into *chunks* similar to the generalized rules learned by explanation-based learning systems. Tambe and Newell found that increasing numbers of "expensive" chunks increase total match time and eventually degrade performance. Chunks become expensive due to an increased number of conditional elements, an increased number of objects that can match these elements, and suboptimally ordered conditional elements.

Instead of monitoring the cost and benefits of rules, Tambe and Rosenbloom [1989] suggest restricting the expressiveness of the learned rules so that the complexity of the match is kept linear in the number of matching conditions. Results of using this technique within SOAR indicate that a greater number of less expressive rules are needed to attain the generality of the more expressive rules, but the match cost is no longer exponential. However, the results are unclear on whether an exponential number of simpler rules will be needed to achieve the

generality of the more expressive rules. Also, the trend toward generating more specific instances of the general rules seems contradictory to the purported benefits of analytical learning.

2.3.3 EGGS

Despite the aforementioned evidence for degrading performance, other analytical learning systems demonstrate improved performance without concern for the number or form of the learned rules. Looking at systems by O'Rorke [1987] and Shavlik [1988], Mooney [1989] uncovers the reasons for these contradictory results. The performance element for Mooney's experiments is the EGGS system [Mooney and Bennett, 1986], which includes a Horn-clause theorem prover and standard explanation-based learning techniques [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] for generalizing proofs.

Mooney's experiments with EGGS reveal that limited use of the learned rules provide better problem-solving speed than full use. Because Shavlik constrains the proofs to be no longer than a specified depth bound, his system makes only limited use of the learned rules (i.e., only those rules that require limited chaining).

Mooney also demonstrates that using a breadth-first search for theorem proving, instead of depth-first, forces limited use of learned rules. Learned rules that require deep sub-goaling to reach a solution are circumvented by the simultaneous consideration of proofs from the original domain theory. The use of breadth-first search in O'Rorke's system accounts for much of the favorable performance. Mooney concludes that limited use of learned rules is advisable until the system has learned the rules necessary to solve the more common problems.

2.3.4 Analytical Performance Response

Although experimentation with the above analytical learning systems confirms the existence of the utility problem, the experiments typically do not show the performance response of the system.[4] This section plots the performance response of a simple analytical learner in Figure 2.12. The analytical learner consists of a forward-chaining planner and a STRIPS-like plan generalizer [Fikes *et al.*, 1972]. Two domains are used in the experimentation: blocks and robot. The blocks domain consists of four operators for stacking and unstacking blocks. The

[4]Cohen [1990] plots analytical learning response curves for several planning domains; however, the curves reflect the performance of learning problem-solver control rules, not macro-operators.

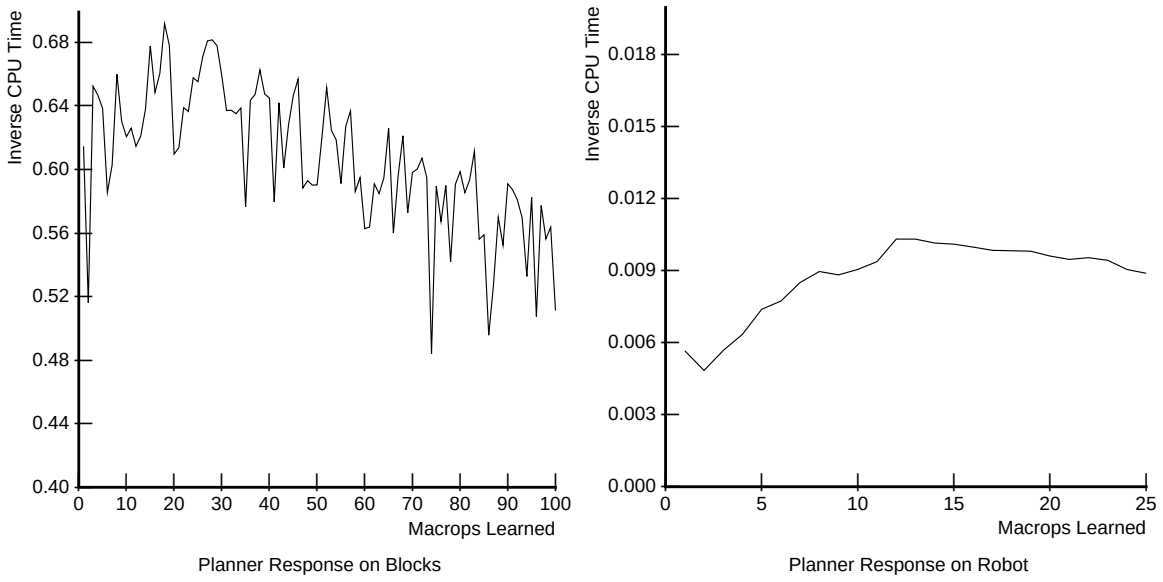

Figure 2.12: Planner performance response.

robot domain consists of eight operators allowing the robot to move boxes within a layout of connected rooms. See Appendix A for a more detailed description of these domains.

The experiments proceed by solving a training problem in the domain, generalizing the resulting plan, adding the generalized plan to the set of available operators, and then measuring the amount of CPU time needed to solve a separate set of test problems using the augmented set of operators. A generalized plan is called a macro-operator, or macrop. Adding a macrop plan to the set of operators increases the planner's control knowledge about how to search the space of possible plans. Therefore, the x-axis of the performance response is the number of learned macrops. The y-axis measures the inverse CPU time needed to solve the set of test problems. Inverse CPU time allows an increase along the y-axis to reflect an increase in planner performance.

Figure 2.12 plots the performance response of the planner while learning macrops in the blocks and robot domains. Although erratic in the blocks domain, both response curves follow the trend of the general utility problem as shown in Figure 2.1.

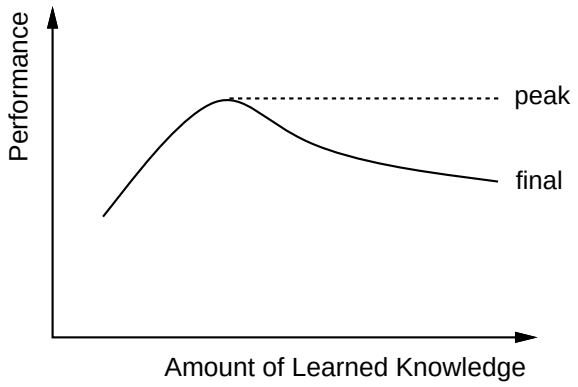

Figure 2.13: Response curve peak and final performance.

2.4 Trends

The previous sections of this chapter verify the existence of the general utility problem in several machine learning methods. Furthermore, the performance responses of these methods follow the general trend illustrated in Figure 2.1. Adopting this trend as a model of the performance response permits the control of the general utility problem by constraining the amount of learned knowledge to reside at the point corresponding to the peak performance.

Tables 2.1 and 2.2 quantify the possible performance gains by using this model-based control of the amount of learned knowledge. Each entry in the tables is the percentage final performance of peak performance averaged over ten performance response curves (see Figure 2.13):

$$\frac{\text{final}}{\text{peak}} \times 100$$

Table 2.1 lists entries for several of the previously described empirical learning methods on five different domains. Table 2.2 lists entries for the Planner analytical learner from Section 2.3.4 on two domains. Note that the entries in Table 2.2 can be arbitrarily deflated by allowing the analytical learner to acquire more macrops. Accompanying each entry is the statistical significance of the difference between the peak and final performance, i.e., the probability that the difference is due to chance fluctuations in the data.

As shown in Tables 2.1 and 2.2, the final performance is less than the peak performance for all but one case. A majority of the differences are statistically significant, and in the cases where the significance is low (table value is high), the peak of the performance response is no worse than the final performance. Thus, the ability to constrain the amount of learned

Table 2.1: Percentage final performance of peak performance for empirical learners on five domains. Statistical significance of difference (peak - final) shown in parentheses.

	Domain				
Method	Breast Cancer	Flag	Flare	Voting	DNF2
ID3	91.2(0.001)	88.2(0.001)	95.0(0.000)	97.6(0.011)	93.6(0.000)
ID3 Chi 99.0	89.0(0.001)	88.5(0.002)	94.4(0.000)	98.1(0.021)	94.4(0.000)
ID3 Chi 99.9	90.8(0.001)	89.9(0.001)	96.1(0.001)	97.0(0.000)	97.2(0.064)
ID3 Reduced-Error	98.6(0.571)	95.4(0.172)	98.7(0.281)	99.7(0.686)	100.3(0.748)
PLS1 $t_\alpha = 0.0$	87.5(0.000)	84.4(0.000)	95.8(0.000)	98.3(0.038)	91.5(0.000)
PLS1 $t_\alpha = 1.0$	87.9(0.000)	96.3(0.232)	97.7(0.000)	98.1(0.050)	92.8(0.000)
PLS1 $t_\alpha = 1.5$	92.4(0.021)	97.6(0.469)	98.5(0.186)	98.9(0.046)	92.8(0.000)
PLS1 $t_\alpha = 2.0$	94.6(0.033)	98.4(0.647)	98.5(0.218)	99.3(0.078)	95.6(0.000)
BP4	82.8(0.287)	89.8(0.004)	88.2(0.711)	92.6(0.450)	91.1(0.141)

Table 2.2: Percentage final performance of peak performance for Planner on two domains. Statistical significance of difference (peak - final) shown in parentheses.

	Domain	
Method	Blocks	Robot
Planner	67.4(0.026)	76.1(0.165)

knowledge to the point corresponding to peak performance will improve the performance of the learner. Although individual methods exist for alleviating the general utility problem in each particular learning method, the performance response model offers a general method for avoiding the general utility problem in many machine learning methods. Chapter 3 uses the performance response model as the basis of an adaptive control approach for maintaining the utility of learned knowledge.

2.5 Analysis

The previous sections of this chapter empirically demonstrate the existence of the general utility problem trend in several learning methods. This section provides a more formal understanding of the mechanisms that cause this trend in the performance response. Section 2.5.1 analyzes empirical learning methods, and Section 2.5.2 analyzes analytical methods. In both cases, the performance response trend results from two contributing forces and depends on a precise definition of the amount of learned knowledge in terms of the generality of this knowledge.

2.5.1 Empirical Learning

The following two sections analyze the performance response and the amount of learned knowledge as they relate to the general utility problem trend in empirical learning. Analysis shows that if the amount of learned knowledge corresponds to the complexity (specificity) of the induced hypothesis, then the performance response trend results from two components affecting accuracy: accuracy on the training data and accuracy on the testing data.

2.5.1.1 Performance Response

The CART program (Classification and Regression Trees) developed by Breiman *et al.* [1984] is another splitting method for inducing decision trees similar to ID3 and PLS1. The emphasis of this treatment of CART is not the details of the method, but a statistical analysis of the performance response (see appendix to Chapter 3 in [Breiman *et al.*, 1984]). Breiman *et al.* show that the shape of the performance (accuracy) response is the result of a tradeoff between bias and variance. *Bias* expresses the degree of fit of the decision tree to the classification surface (training instances). A low bias (many small hyper-rectangles) is preferred to a high bias (few

large hyper-rectangles), because low bias allows a more precise fit to the data. However, a low bias increases the likelihood that hyper-rectangles produce classification errors due to a majority of the wrong class. Breiman *et al.* refer to this source of classification error as *variance*. This is not variance in the statistical sense of the expected value of the squared error, but an estimate of the discrepancy of the classification error from the Bayes error.

The analysis expresses the bias and variance in terms of the number of leaves L in the decision tree. Assuming binary splits at each node of the tree, the number of splits is $L - 1$. Therefore, the behavior of the bias and variance as the number of splits increase will be similar to the behavior as L increases. The expression for the classification error $R(L)$ in terms of the bias $B(L)$ and the variance $V(L)$ is

$$R(L) = B(L) + V(L) + R^* \tag{2.1}$$

where R^* is the Bayes optimal classification error. Breiman *et al.* derive the following constraints on the bias $B(L)$ and the variance $V(L)$:

$$B(L) \leq \frac{C}{L^{2/M}}, \quad V(L) \leq \sqrt{\frac{L}{N}}, \quad V(L \simeq N) \leq R^*$$

where C is a constant, M is the dimension of the instance space (i.e., number of features used to describe the training instances), and N is the number of training instances. Note that these expressions are for the classification error. As predicted, the bias decreases rapidly for small L and more slowly as L increases. The variance increases slowly as L increases. When $L \simeq N$ and each hyper-rectangle contains one training instance, the variance is bounded by the Bayes error R^*.

Equation 2.1 is an expression of the classification error response curve. Figure 2.14a plots the bias $B(L)$, variance $V(L)$, Bayes error R^*, and estimated classification error $R(L)$ from Equation 2.1, where $C = 0.35$, $M = 20$, $N = 1000$ and $R^* = 0.15$.[5] The plot extends from $L = 0$ to $L = N = 1000$; however, the stopping criteria of actual decision tree induction programs would discontinue splitting at a point much less than N. For comparison to previous response curves, the error response curve is subtracted from one to yield the accuracy response curve in Figure 2.14b. The similarity of this performance response to that of Figure 2.1 supports

[5] For binary decision trees, $L \leq 2^M$.

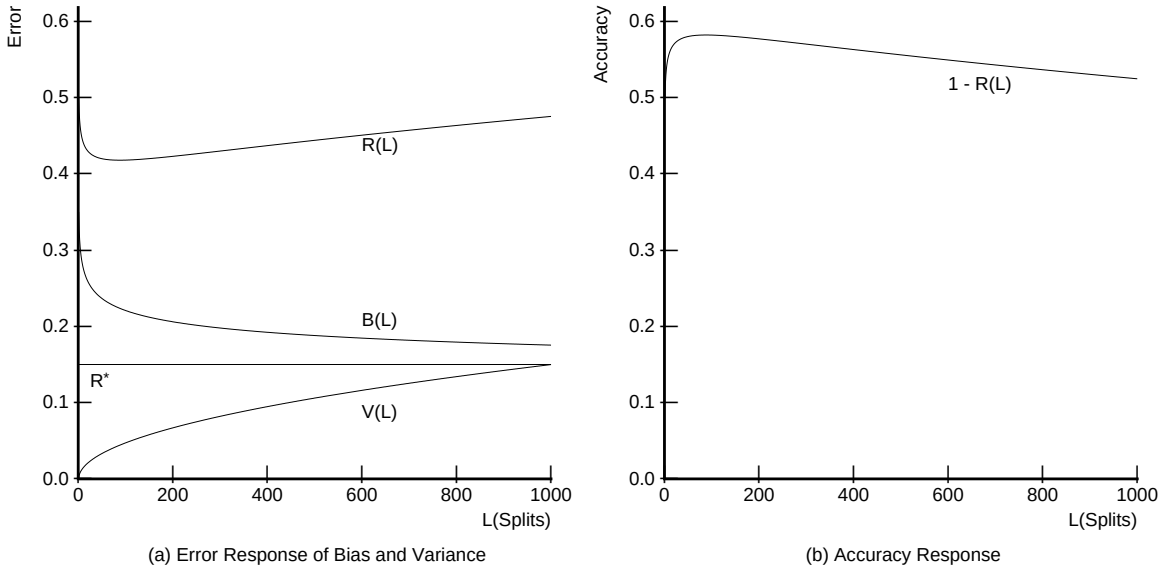

Figure 2.14: Performance response curve derived by Breiman et al. [1984] for a decision tree induction algorithm.

the existence of a single peak and the inevitability of overfit in splitting algorithms without appropriate stopping criteria or post-pruning techniques. Maximizing performance while avoiding overfit requires the determination of the number of splits L corresponding to the peak of the performance response.

A similar analysis applies to agglomerative methods. Each time a splitting method makes a split in the decision tree, the resulting DNF expression of the hypothesis replaces a single disjunct with two, more specific disjuncts (assuming binary splits). Therefore, adding a disjunct to the DNF hypothesis in an agglomerative method is analogous to making a split in a splitting method. The above definitions of bias and variance apply directly to the agglomerative case. Decreasing the bias increases the number of disjuncts until each disjunct describes a single training instance. Variance, the error due to incorrect classifications made by the disjuncts on unseen testing instances, increases with decreasing bias (see the discussion of small disjuncts in Section 2.2.2.2). The corresponding expressions for bias and variance as a function of the number of disjuncts have a similar behavior as those depending on the number of splits, and the agglomerative performance response follows the behavior in Figure 2.14.

The performance response trend in neural networks is also the sum of the performance on training data and the performance on testing data. Before relating performance to the number

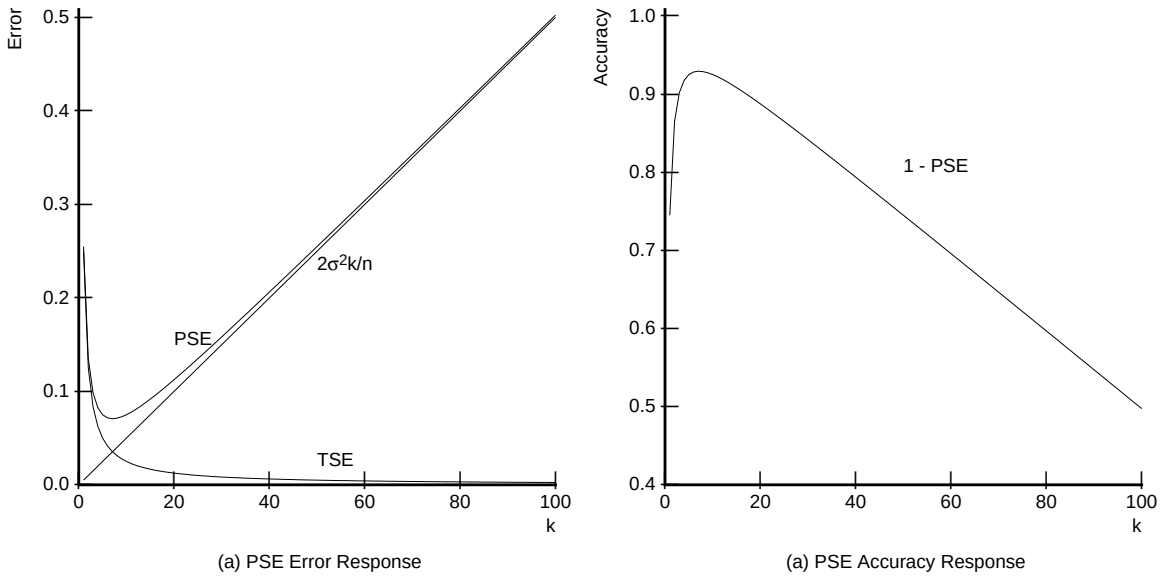

Figure 2.15: Performance response curve derived by Barron [1984] for a network as a function of the number of coefficients k in the network model.

of cycles, this analysis first considers the number of coefficients in the model represented by the network. Network models with increasing complexity (e.g., number of hidden units) have higher numbers of coefficients. If the complexity of the network is higher than the complexity of the problem, the complex network will use the overabundance of coefficients to overfit the training data.

Barron [1984] derives an expression for the predicted squared error (PSE) of the network that depends on the number of coefficients. The expression for PSE is

$$\text{PSE} = \text{TSE} + 2\sigma^2 \frac{k}{n}$$

TSE stands for the squared error of the network on the training examples, σ^2 is a prior estimate of the true error variance, k is the number of coefficients in the network model, and n is the number of training examples. One estimate of the true error variance σ^2 is the actual variance in the training data. The second term of PSE serves as an overfit penalty for excessively complex models. Assuming TSE has a similar behavior as the bias in Figure 2.14, Figure 2.15a plots the two components of PSE and their sum as a function of k for $n = 100$. Figure 2.15b plots the same function subtracted from one to show the same orientation of previous performance responses. The resulting curve confirms the general utility problem trend in networks.

The above analysis uses the number of coefficients k as a measure of the amount of learned knowledge; however, the performance responses for neural networks use the number of cycles as a measure of the amount of learned knowledge. One possibility for relating the number of cycles to the number of coefficients k is to show that the higher numbers of coefficients in the network model are not used (negligibly small) until later cycles. In other words, earlier cycles use fewer coefficients to learn global patterns in the training data. As the cycles continue, the network attempts to reduce the error on noisy (or anomalous) training data by utilizing more coefficients to fit a higher-degree function to the training data.

The following argument derives from our observations of the error back-propagation method during the course of learning. The observations reveal that the network quickly learns to correctly classify a majority of the training data and uses the remaining cycles to learn a smaller subset of the training data. One cycle involves a single pass through the entire set of training data, where each incorrect classification initiates the error back-propagation procedure to update the weights toward correcting the error. Initially, a majority of the weight updates are due to errors on the training data representing the global patterns (the more prevalent data). After the network learns these global patterns, the majority of weight updates are due to errors on less prevalent patterns in the training data. One possible interpretation of this behavior is that later cycles attempt to fit higher degrees of the function represented by the training data. If this interpretation holds[6], then as the number of cycles increases, so does the degree (complexity) of the hypothesis learned by the network. Therefore, roughly similar behavior to that of Figure 2.15 will exist if the number of cycles replaces k along the amount of learned knowledge axis.

2.5.1.2 Amount of Learned Knowledge

The previous section shows how number of splits, number of disjuncts and number of cycles are appropriate definitions for the amount of learned knowledge in splitting, agglomerative, and network learning methods, respectively. These definitions are appropriate, because each is an instance of a more general definition expressing the amount of learned knowledge as the degree of complexity (specificity) of the learned hypothesis. Figure 2.16 shows the hypothesis

[6]Observations by Mozer and Smolensky [1989] support a similar interpretation, but more experimentation is necessary to confirm the reason for this behavior.

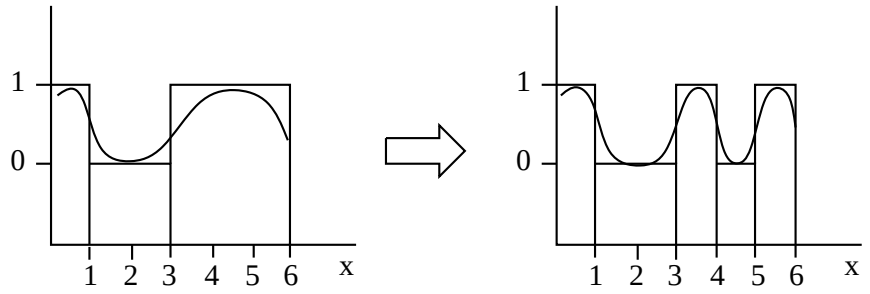

Figure 2.16: The degree of the function corresponding to the learned hypothesis before and after a transformation specializing the hypothesis. The more specific hypothesis has a higher functional degree (6) compared to the less specific hypothesis degree (4).

as a function over a one-dimensional instance space. The function on the right is a result of learning that the data in the region between x=3 and x=6 in the function on the left does not always return 1. Refining this region specializes the hypothesis and increases the degree of the corresponding function from four to six.

The change in Figure 2.16 has analogs in the three transformations (split, disjunct, cycle) for the three empirical learning methods. The transformations result in a more specialized hypothesis and a higher degree function corresponding to this hypothesis. Therefore, the definition of the amount of learned knowledge for a learning method implies an ordering of these transformations from general to specific, or lower degree to higher degree of complexity. Given that the hypothesis has a sufficient degree of complexity to allow overfit, this transformation order insures the presence of the general utility problem trend in empirical learning. Section 3.4 discusses this issue further.

Referring back to the PLS1 performance response plots in Figure 2.8, each row of plots corresponds to a difference value of t_α. Increasing t_α decreases the number of splits made by PLS1. This observation suggests that t_α is a possible candidate for the amount of learned knowledge in PLS1. An increase along the amount of learned knowledge axis would correspond to a decrease in t_α. More generally, any learning parameter that constrains the number of splits (e.g., the confidence parameter in ID3's chi-square pre-pruning) is a candidate for the amount of learned knowledge. The change in t_α along the amount of learned knowledge axis must be small enough to perceive the general utility problem trend in the performance response. The performance value for each t_α value is the final performance value shown in the plots of Figure 2.8 corresponding to the same t_α value. Figure 2.8 shows that using values for t_α in the

recommended range $(1.0 - 2.0)$ does not capture the peak of the performance response. Higher values of t_α are necessary to perceive the peak. Therefore, not only must the changes in t_α be small, but the initial value of t_α must be large enough to prevent any overfit in order to perceive the peak.

2.5.2 Analytical Learning

An analytical learner is similar to an empirical learner in that both seek a concept that maximizes performance. The concept sought by an analytical learner is a set of macro-operators minimizing the time taken by the problem solver to solve problems from some domain. If the set of problems used to train the learner is irrepresentative of the distribution of problems in the domain, then the performance obtained for the training examples may degrade performance on the testing examples for reasons similar to overfit in empirical learners. However, a more detailed look at analytical learners reveals that the factors underlying the performance degradation are different from the factors affecting empirical learners. This section considers these factors in more detail by analyzing the performance response and the amount of learned knowledge as they relate to the general utility problem trend in analytical learning.

The analysis pertains to analytical learning methods that acquire macro-operators (macros) composed of the individual operators used to solve a problem. As with empirical learning, the analytical performance response is the result of two contributing factors. However, the factors affecting problem-solving time differ from those affecting classification accuracy. One factor is the decrease in problem-solving time due to solving a problem with a macro instead of the original operators. The second factor is the increase in problem-solving time due to the cost of retaining the macro. One constraint from this analysis is an ordering of the amount of learned knowledge from general to specific. General macros apply to more problems and have greater benefit than more specific macros. Therefore, the analytical learner should acquire more general macros before more specific ones to insure an initial increase in the performance response. In this scenario, the performance response for analytical learning follows the general utility problem trend illustrated in previous sections.

2.5.2.1 Performance Response

The analytical learner analyzed in this section consists of a list of original operators O, a list of macros M, and a problem solver. Given a problem, the problem solver uses M and O to derive a solution to the problem. The analytical learner generalizes the solution to form a new macro m and adds m to the end of the list M. Performance of the problem solver is the time to solve a set of test problems.

Minton [1990] identifies three ways in which macro-learning affects the problem-solving performance of an analytical learner. First, adding macros to the list of operators changes the traversal order of the search space. Macros try paths through the search space earlier than normal in the hopes that the problem can be easily solved with a macro without resorting to the original operators. Second, the availability of macros reduces the cost of searching some paths in the search space. In addition to the savings of solving problems directly, macros can also solve subgoals within the search space. Third, macros introduce redundancy in the search space by visiting states that will be visited later in the search space and by testing operator preconditions that will be tested again later. The first two effects generally improve problem-solving performance, and the third effect degrades performance.

Analyzing these effects in more detail yields the behavior shown in Figure 2.17. Each new macro decreases the time necessary to solve the problem (and similar problems) generating the macro. The time to solve the problem without a macro is approximately r^d, where r is the number of operators available to the problem solver, and d is the difficulty of the problem in terms of the depth of the solution in the search space. Replacing this exponential search with a lookup in the list of macros decreases the amount of problem-solving time by approximately r^d. In addition, the learned macro may reduce the search space for other problems by solving subgoals in one step that previously took more than one step. The extent of the decrease in time depends on whether the learned macro is general or specific. If the macro is specific, then the decrease in time will be small, because the macro will apply to few other problems and to few subgoals in the search space. If the macro is general, then the decrease in time will be greater, because the macro applies to more problems and search-space subgoals. In order for the performance response behavior to match the previously-observed behavior, the decrease in time due to earlier macros must be greater than the time decrease for later macros. Therefore,

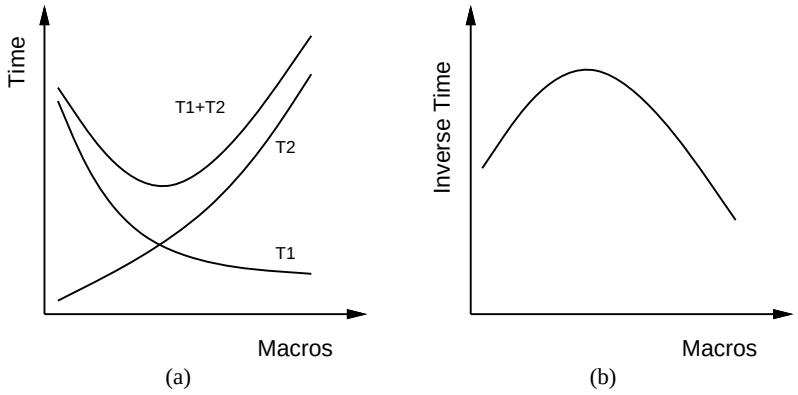

Figure 2.17: The behavior of the analytical performance response involves two factors represented by the curves T1 and T2. The T1 curve represents the first two factors described by Minton [1990], and the T2 curve represents the third factor. The resulting problem-solving time performance (a) yields the general utility problem trend for inverse problem-solving time (b).

the system must acquire macros in the order from general to specific. If the system meets this constraint, then the factors decreasing problem-solving time will have the behavior of the T1 curve in Figure 2.17a.

The addition of macros also has a detrimental effect on performance. As the system learns more macros, the problem-solver includes these macros along with the original operators for solving problems. The increase in available operators causes an increase in the branching factor of the search space from r^d to $(r+m)^d$, where m is the number of added macros. However, at the same time, the capability of the macros to reduce the search for subgoals reduces the difficulty of the problems (i.e., reduces d). Assuming the general-to-specific ordering constraint is in effect, the increase in problem-solving time will be small for the early, more general macros due to the reduction in the subgoal search. Also, since more general macros have fewer preconditions, the problem-solver spends less time matching the preconditions of these macros. The result is an exponential increase in problem-solving time with a slow initial rate of increase that grows with the acquisition of later, more specific macros. The T2 curve in Figure 2.17a depicts this behavior.

Figure 2.17a also shows the sum of the two curves T1 and T2. For comparison to the previous experimental results, Figure 2.17b shows the inverse of the curve from Figure 2.17a. This performance response relates inverse problem-solving time to the number of learned macros.

Assuming the analytical learner acquires macros in the order from general to specific, the performance response follows the general utility problem trend.

2.5.2.2 Amount of Learned Knowledge

The analysis of the previous section shows that the number of macros is an appropriate measure of the amount of learned knowledge for perceiving the general utility problem trend in analytical learning methods. The analysis constrains the use of this measure such that the system acquires macros in a general-to-specific order. However, the analytical performance responses shown in Section 2.3.4 do not constrain the macro-acquisition order in this way. The effect of ignoring the order is the possibility of an initial decrease in the performance response before the more global increase depicted in the previous section. The initial decrease is due to the possibility of acquiring a few specific macros before a general macro. Since example problems generating more general macros will be more frequent in the set of training problems, there is a greater probability of learning a general macro, and the initial decrease in the performance response will be only temporary. Both of the experimental performance responses in Figure 2.12 exhibit this behavior.

Analysis of both analytical and empirical methods verify the commonality of the performance response trend. The analysis indicates that the trends result from two factors: one increasing performance and one decreasing performance. Furthermore, both analyses constrain the order of increasing the amount of learned knowledge to be from general to specific. The overall behavior of the performance response is a curve increasing rapidly to a single peak and then decreasing more slowly after the peak. The next chapter uses a model of this behavior as the basis for an adaptive control approach for maintaining the utility of the knowledge acquired by learning methods.

Chapter 3

Model-Based Adaptive Control

The model-based adaptive control (MBAC)[1] approach uses the trend identified in Chapter 2 as a model to control the amount of learned knowledge in order to maintain utility. The model describes the performance response for a particular task domain (e.g., Flag or DNF2), performance dimension (e.g., accuracy or speed) and knowledge transformation (learning method). MBAC's model of the performance response is a parameterized curve. MBAC fits the curve according to previously observed samples from the actual performance response. Using this instantiated model of the performance response curve, MBAC determines the point on the curve having the desired level of performance and recommends learning the amount of knowledge corresponding to this point.

The proposed MBAC approach resembles an adaptive control loop as shown in Figure 3.1. First, the performance element uses the knowledge to perform some task. MBAC compares the performance on the task to the user-defined performance objectives. This performance comparison serves as feedback to improve the model's estimate of the true performance response. Using the updated model, MBAC decides how to transform the knowledge in order to achieve and maintain the desired performance objectives.

For example, Figure 3.2 shows the response of a splitting algorithm on a set of test data as the algorithm learns using a separate set of training data. Assuming no splits have been made, and Model 1 is the current instance of the model, the control decision would be to make

[1] Model-based adaptive control is a new term similar to the term model-reference adaptive control used in adaptive control theory [Sastry and Bodson, 1989]. The modification of the new term serves to distinguish it from the more formally defined adaptive control counterpart.

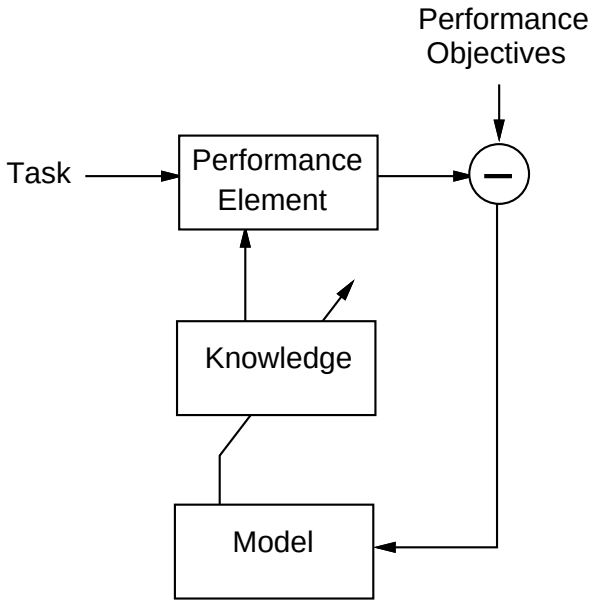

Figure 3.1: Adaptive control of learning.

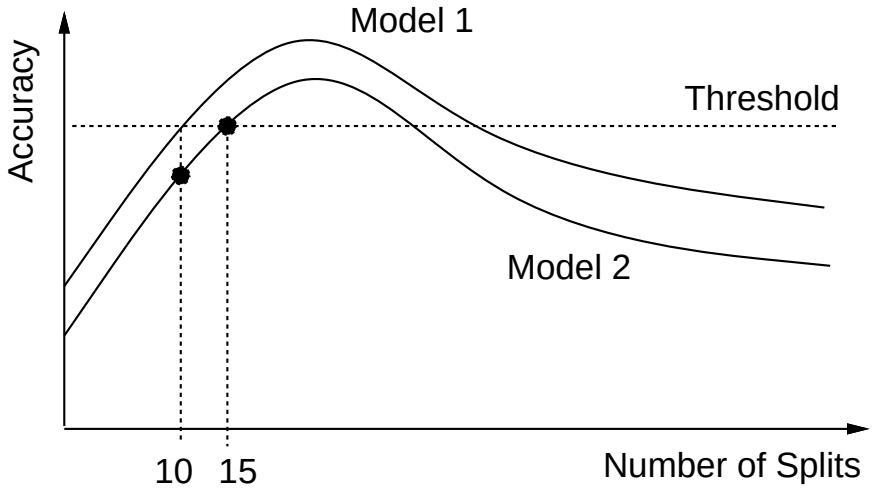

Figure 3.2: Two models of the performance response of a splitting algorithm.

ten splits to achieve the desired threshold. However, when the resulting performance is still beneath the threshold, MBAC uses the new performance data for updating the model to Model 2. This model then decides to make five more splits to achieve the threshold. Assuming Model 2 is the correct model, performance on the task will meet the threshold.

The important components of the MBAC approach in Figure 3.1 involve the knowledge, the performance objectives, the model, and the transformations performed on the knowledge as suggested by the model. Section 3.1 defines the components, and Section 3.2 outlines the MBAC algorithm. The remaining sections discuss the issues involved in the design of the components. Chapter 4 describes specific implementations of these components used for experimentation.

3.1 Definitions

This section defines the MBAC approach. The definitions clarify the scope of the approach and provide a precise context in which to discuss the issues of the approach. The definitions and accompanying examples follow the diagram in Figure 3.1 and the example of Figure 3.2.

Definition 1 A *task* T consists of a set of problems P_T and a set of performance objectives $O_T = \{o_1, \ldots, o_n\}$.

For example, the task may be the classification of a set of examples with 95% accuracy. P_T would be the set of examples, and O_T would contain one element corresponding to the accuracy performance objective. A subset of the examples in P_T comprise the training set serving as the initial knowledge. The remaining examples in P_T comprise the testing set used by the performance element to evaluate the knowledge.

Definition 2 A *performance objective* o_i is a pair $\langle d_i, t_i \rangle$ where the performance dimension d_i is a quantity measured during the execution of the performance element, and the performance threshold t_i is the desired value of the performance dimension.

The sample task above has one performance objective whose dimension is accuracy and whose threshold is 95%. MBAC measures accuracy as the ratio of correctly classified examples in the test set to the number of examples in the test set. If the performance dimension is CPU time, MBAC would measure this dimension as the amount of CPU time needed by the

performance element to solve the examples in the test set. If the user desires peak performance, the threshold can be set to 100% (or infinity when there is no maximum value). Section 3.5 discusses performance objectives in more detail.

Definition 3 The *knowledge* K is a possibly more general expression of the training examples in P_T. The expression is in a form usable by the performance element and modifiable by knowledge transformations.

The definition of knowledge is unavoidably abstract, because the knowledge may take several forms depending on the task. For example, the knowledge may be in the form of a decision tree, a network, or a set of problem-solving macro-operators. As described in Section 3.3, MBAC uses a hybrid knowledge representation that maintains separate knowledge structures for each transformation. The knowledge available to the performance element is the most recently transformed knowledge for the task. For example, if the most recent control decision was to perform some number of splits on a decision tree representation, the knowledge available to the performance element during the next evaluation would be the resulting decision tree.

Definition 4 The *performance element* PE uses the knowledge K to solve a set of problems $P \subseteq P_T$ from task T. PE produces a vector of values representing the measured performance dimensions for the performance objectives O_T associated with task T.

In our example, P would be the set of test examples from P_T. The performance element would return a one-dimensional vector consisting of the accuracy of the knowledge in classifying the set of test examples. MBAC treats the performance element as a "black box". As long as the input and output requirements are maintained, the performance elements are interchangeable.

Definition 5 A *knowledge transformation* KT is a pair of methods $\langle KT^-, KT^+ \rangle$ for decreasing and increasing the amount of learned knowledge in K. Each method has an associated cost function c^-, c^+ representing the resource cost of performing the knowledge transformation.

One possible knowledge transformation mentioned above is the splitting method. For this transformation, KT^+ is the execution of a single split in the decision tree, and KT^- is the removal of the most recent split. One expression of the cost functions would be the average transformation cost of previous transformations. Section 3.4 discusses the issues involved with knowledge transformations.

Definition 6 A *model* M is a parameterized function relating the value of the performance dimension d for task T to the amount of learned knowledge as varied by knowledge transformation KT. Associated with the model is the certainty δ_M in the prediction of the value of d from the amount of learned knowledge.

MBAC uses a parameterized model to fit the expected trend of Figure 2.1. An example of a model would be the relationship between accuracy and the splitting transformation for a particular task. An instance of a model is a particular choice for the parameters of the model. A model type is a particular parameterized function relating performance to the amount of learned knowledge. Unless otherwise noted, further discussion uses the terms *model* and *model instance* interchangeably. The certainty of the model may depend on the certainty of the parameters or the deviation between the model and actual performance data. Section 3.6 discusses properties of MBAC models.

Definition 7 *Model-Based Adaptive Control* (MBAC) maintains a model for each ⟨task, knowledge transformation, performance dimension⟩ triple. MBAC updates the parameters of the models according to the vector of performance dimension values produced by the performance element. MBAC uses the updated models to select a knowledge transformation for achieving the performance objectives of a task. MBAC then transforms the knowledge and re-executes the performance element on the task.

MBAC maintains several models which compete for the opportunity to transform knowledge in order to achieve the performance objectives. Each model is an instance of the same type (parameterized curve). Actual performance measurements provide data for refining the models and improving the transformation decisions. The next section describes MBAC's adaptive control algorithm in more detail.

3.2 Adaptive Control Algorithm

Figure 3.3 outlines the MBAC algorithm underlying the block diagram of Figure 3.1. Given a task, a set of performance objectives for the task, and a set of knowledge, the MBAC approach begins by executing the task using the performance element and the current knowledge. During task execution MBAC measures the performance dimensions associated with the given

```
Given: task
       performance objectives
       knowledge K

Repeat
   Evaluate performance of K on task
   Identify models for task and performance objectives
   Update models based on performance feedback
   Use models to predict certainty of attaining thresholds
   Select transformation minimizing uncertainty and cost
   Apply transformation to K
Until all performance objectives satisfied
```

Figure 3.3: Model-Based Adaptive Control algorithm.

performance objectives. After task execution, MBAC collects all model instances pertaining to the task and each of the performance dimensions. For each performance dimension MBAC may retrieve multiple model instances, one for each knowledge transformation applicable to the task and performance dimension.

Next, MBAC uses the performance measurements to update the models. The performance element uses knowledge associated with the most recently applied transformation; therefore, MBAC updates only those models associated with this transformation. Assuming the current knowledge does not satisfy one or more of the performance objectives, each model pertaining to a dimension of an unsatisfied performance objective suggests a transformation for achieving the objective and estimates the transformation's certainty of success. The success of the transformation depends on the ability to achieve unsatisfied objectives and preserve already satisfied objectives.

MBAC then selects the transformation maximizing the estimated certainty of success and minimizing transformation cost. The estimated certainty of success combines the model's estimate of attainable performance with the certainty of the model. MBAC applies the transformation to the associated knowledge, installs this knowledge as the current knowledge for solving the task, and re-executes the performance element on the given task. This process continues until the current knowledge satisfies all performance objectives on the task.

The following sections discuss the issues involved in the MBAC approach. The definition of knowledge in Section 3.1 is imprecise and does not address the representation issue of compatibility with the performance element and individual knowledge transformations. Section 3.3 discusses MBAC constraints on knowledge compatibility and transformation. Section 3.4 considers the issues involved in decomposing current learning methods into knowledge transformations. MBAC uses explicit performance objectives to provide performance feedback to the control loop. Section 3.5 discusses tradeoffs among performance objectives, unachievable objectives, and the MBAC algorithm's stopping criterion of performance objective satisfaction. Finally, Section 3.6 considers several model-related issues: validity, MBAC constraints, model type, identification and transformation selection. Chapter 4 describes specific implementations of these components of the MBAC approach.

3.3 Knowledge Representation

The MBAC approach requires the knowledge to be compatible with the performance element and the knowledge transformations. This section discusses a hyrid knowledge representation that meets these requirements.

3.3.1 Compatibility

The knowledge acquired by learning systems may take a variety of forms. For example, the learning systems discussed in Chapter 2 acquire knowledge in the form of decision trees, decision lists, neural nets, and planning operators. These examples represent only a subset of the set of knowledge representations used by learning systems. Despite this variety in knowledge representation, the MBAC approach attempts to unify learning methods by exploiting the similarity in their performance response curves. However, the independent variable in these curves (the amount of learned knowledge) has different units for each learning method. For example, performance response curves for neural nets have units of number of cycles; whereas, curves for splitting methods have units of number of splits. Because the MBAC approach controls multiple models recommending different transformations to different forms of knowledge, the approach must execute these differing knowledge transformations, while maintaining the ability to communicate the different knowledge representations to the performance element.

Solutions to this issue fall along a spectrum. One end of the spectrum corresponds to using a single knowledge representation and modifying the knowledge transformations to work on this representation. The main drawback to this approach is the difficulty of modifying transformations that are intimately related to the representation of the transformed knowledge. For example, the effect of a splitting transformation on a network representation, or even a set of rules not oriented in a decision tree, is unclear. Likewise, the effect of n cycles' worth of learning in a neural net method on a decision tree representation is also unclear. The effect of a transformation on the representation for which it is designed is the main motivation for choosing one transformation over another. Removing the dependence of the transformation on its representation would reduce the transformation's ability to produce changes in performance.

The other end of the spectrum corresponds to maintaining different knowledge representations for the different types of knowledge transformations and modifying the performance element to work with these multiple representations. Although the MBAC approach is dependent upon the performance element for evaluating performance objectives, one of the benefits of the approach is the ability to adapt the knowledge to changes in the performance element. Therefore, one drawback to the multiple representation approach is that the performance element must handle multiple representations, which places a burden on the designer, especially when the need arises to incorporate new representations. Even if the performance element can handle multiple representations, a second drawback is that the control element must choose which representation to use for the current problem.

The ideal point along the spectrum falls in the middle. One would like to preserve the multiple representations for the sake of the differing transformations, while allowing the performance element to deal with a single representation. One solution is the use of production rules with procedural augmentations for handling the underlying representations. For example, the production rule for predicting the class of an object from its features may look like

$$\text{feature1(OBJECT,?v1)} \wedge \text{feature2(OBJECT,?v2)} \wedge \cdots \longrightarrow \text{class(OBJECT,?c)}.$$

The version of the rule for a decision tree representation would instantiate the class variable `?c` according to the class of the leaf node at the end of the decision-tree path traversed according to the feature values `?v1, ?v2, ....` The neural net version of the rule would put the feature values at the input units of the network and instantiate the class according to the values

at the output units. Using this procedural rule representation as an intermediate between the performance element and the multiple representations requires the performance element to handle only one representation, and allows the knowledge transformations to modify the knowledge representation for which they were designed.

The intermediate representation allows the possibility of multiple rules applying to a single problem. For example, in determining the class of the above object, both the decision tree rule and the neural net rule can predict (possibly different) conclusions. When multiple representations are applicable, MBAC determines which rule (i.e., representation) to use for solving the problem according to the most recent knowledge transformation. The next section discusses the method for deciding among different representations.

3.3.2 Transformation

The transformation of knowledge is a two-stage process. First, MBAC selects a knowledge transformation to perform on the knowledge associated with that transformation. Second, MBAC replaces the knowledge currently used to solve the task with the transformed knowledge. For example, assume the current task is the classification of some object, and the current knowledge representation is a neural network. After the performance element attempts the task using the network, MBAC updates the models for the task and the neural network transformation (cycle/uncycle). Assuming some unsatisfied performance objectives, MBAC may decide that the split/unsplit decision tree transformation has a better certainty of achieving the performance objectives. MBAC transforms the decision tree for this task according to the model's suggestion and replaces the neural network with the transformed decision tree as the knowledge used for the object classification task.

The most recently transformed knowledge for a task is MBAC's recommendation of the best knowledge that the performance element can use to solve the task. By swapping the knowledge underlying the rule-based representation used by the performance element, MBAC maintains compatibility with both the performance element and the individual knowledge transformations.

3.4 Knowledge Transformations

The MBAC approach makes four demands on a knowledge transformation. First, the transformation must make small changes to the knowledge. Small changes are necessary to avoid oscillation about the performance thresholds and to perceive the performance response trend revealed in Chapter 2. Second, MBAC must execute the transformations in a specific order. The shape of the performance response depends on the application order of transformations. Third, the transformation must be reversible. MBAC controls the amount of learned knowledge, which implies the need to both increase and decrease this amount. Fourth, the transformation must have a measurable cost. MBAC selects transformations based not only on the achievable performance, but also on the expected resource cost of executing the transformation. This section discusses the issues involved with satisfying these requirements.

3.4.1 Granularity

The granularity of a knowledge transformation refers to the extent of the change in performance made by applying the transformation. The motivation for decomposing learning methods into smaller grain-sized transformations is two-fold. First, smaller changes in performance during the course of learning allow a more detailed perception of the performance response. The transformation grain size should be small enough to perceive the trend described in Chapter 2. Figure 3.4 shows three transformations differing in grain size. Transformation T3 has the largest grain size and totally obscures the shape of the performance response. T3 represents the grain size of an entire learning method not decomposed into smaller knowledge transformations. Transformation T2 has a smaller grain size and begins to reveal the performance response, but fails to indicate significant properties such as the true peak. Transformation T1 has the smallest grain size and provides the best perception of the performance response.

Given that one wants to drive the learning to the peak of the performance response, the second motivation for small grain-sized transformations is controllability. The greater precision with which the MBAC approach can control the amount of learned knowledge, the closer the performance may converge to the peak of the performance response. Only the smallest grain-size transformation T1 in Figure 3.4 is able to reach the peak of the performance response.

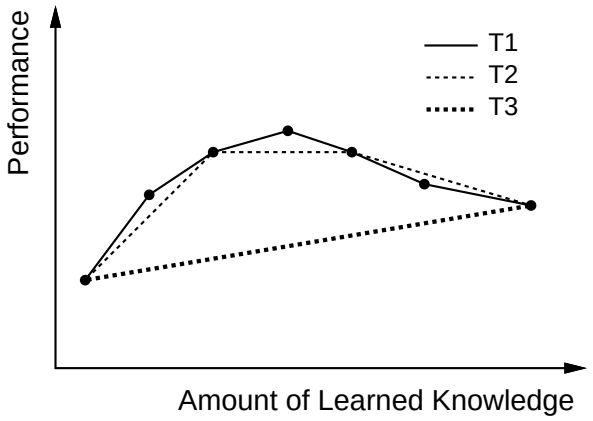

Figure 3.4: The performance responses of three different grain-sized transformations T1, T2 and T3.

Both of the above motivations imply that the granularity should be chosen as small as possible. However, two opposing forces work to increase the transformation grain size. First, the transformation must be complete, which means that the resulting knowledge is in a form equally compatible with the performance element as the original knowledge. Second, smaller grain-sized transformations imply more frequent performance evaluations and more costly model fitting due to the greater number of samples. The increased precision in the perception of the performance response may provide little further information compared to the added cost. Choosing a larger grain size allows less expensive control.

Constraints from the knowledge representation and the learning method typically indicate an appropriate granularity for the knowledge transformation (e.g., one split for a splitting method or n cycles for a neural network method). One way to determine an appropriate level of granularity is to plot sample response curves using the transformation. If the performance response does not resemble the trend of Chapter 2, or the changes in performance are too large compared to the expected changes in performance objectives, then a smaller grain size is recommended.

3.4.2 Order

The performance responses in Figure 2.2 and the analysis in Section 2.5 show that the application order of transformations affects the shape of the performance response. The MBAC approach must perform the transformations in the proper order to insure that the performance

response follows the trend of the general utility problem as in Chapter 2. For empirical learning the trend results from the tendency to overfit the training data as the knowledge moves from general to specific. Given that overfit will eventually occur, avoiding the move to specific knowledge will yield the desired trend. For example, the breadth-first ordering for splitting methods avoids (for as long as possible) the specific (overfitting) knowledge deeper in the decision tree.

The general utility problem trend in analytical learning results from the tendency to learn low-utility knowledge. An analytical learner learns problem-solver search control knowledge in the form of a decision list. Initially, the decision list contains the primitive operators, which are applied in some sequence at each state in the search space. The analytical learner specializes the control knowledge by adding macro-operators to the decision list. General macro-operators result from similar problems having greater frequency in the set of training problems. As the method learns an increasing number of macro-operators, the macro-operators will be more specific to unique training problems, and not a result of general trends in the distribution of problems. Therefore, learning more macro-operators eventually over-specializes the control knowledge and degrades performance (reduces utility) on unseen problems.

The application of transformations should follow an order that avoids specific, low-utility knowledge for as long as possible. The peak of the performance response corresponds to the desired knowledge, and deviations from this knowledge cause performance to fall off from the peak. The direction of the decrease depends on whether the knowledge becomes more general or more specific with respect to the learning task. Ordering such transformations to move knowledge from most general to most specific will yield the desired trend in performance response.

3.4.3 Reversibility

Techniques for reversing a knowledge transformation depend on the properties of the transformation. Some transformations imply a specific ordering, and therefore are easily reversible in place. For example, the splitting transformations of Chapter 2 always split the next node in a breadth-first traversal of the decision tree. Therefore, reversing a split involves removing the last split that was made in a breadth-first traversal.

Other types of transformations are deterministic, but not in-place reversible. A cycle in a neural net method is such a transformation. The cycle is not in-place reversible, because the weight update formula may not be invertible. However, since the cycle is a deterministic

transformation, the transformation can be reversed by restarting from the beginning and performing a lesser number of cycles. More efficient techniques periodically record the state of the knowledge during the course of learning and avoid the need to restart from the beginning.

Non-deterministic transformations can be reversed in a similar way by recording the non-deterministic choices made during each transformation. For example, genetic algorithms make transformations based on stochastic decisions [Goldberg, 1989]. Insuring true reversibility of such transformations requires a complete record of the stochastic choices.

3.4.4 Cost

Associating a cost with each transformation helps MBAC make efficient use of the available resources. Different transformations have different costs. Also, the forward and reverse directions of a transformation may have different costs. Given these costs, MBAC can estimate the total cost of using a transformation to achieve the desired performance objectives. Comparing transformations based on their cost and their ability to achieve desired performance allows MBAC to choose a transformation most likely to achieve the performance objectives using the least amount of resources.

The use of transformation cost in the MBAC approach is not intended to address the general issue of measuring the cost of learning. The MBAC approach incorporates transformation cost estimates in order to trade off performance certainty with available resources. One method for estimating transformation cost is to average actual costs of forward and reverse transformations incurred during previous applications.

Chapter 4 describes the implementation of the transformations used in experimentation with the MBAC system.

3.5 Performance Objectives

The individual machine learning methods described in Chapter 2 seek to obtain optimal performance. However, the MBAC approach converges to specific (possibly non-optimal) performance objectives. Associating multiple explicit performance objectives to each learning task provides two advantages. First, the performance objectives can change, and MBAC will adapt accordingly. Second, MBAC can improve performance along one dimension at the expense of another

less important dimension. Adaptability to change and performance tradeoffs are not possible when multiple performance objectives are combined implicitly within the learning method.

Multiple performance objectives also bring disadvantages arising from the case when not all the objectives are achievable. MBAC must decide whether to achieve some objectives and violate others or reach a performance compromise among all the objectives. This section addresses the issues related to performance objectives.

3.5.1 Adaptability

Optimal performance is difficult to achieve for real-world tasks. The general utility problem results from learning methods that expend too much effort towards reaching optimal performance. One alternative to requiring optimality is to define an acceptable level of performance for the task. The acceptable level of performance may change over time; therefore, learning methods seeking acceptable performance levels must adapt their hypotheses to changes in the performance objectives. MBAC's utilization of explicit performance objectives provides such adaptability.

3.5.2 Performance Tradeoffs

A single performance dimension is not always enough to constrain a learning task. For example, some machine learning methods implicitly define performance objectives for hypothesis simplicity as well as accuracy. The implicitness of the objectives forces the method to overfit when attempting to optimize all the objectives. Defining explicit performance objectives for a learning task allows the MBAC approach to improve performance along one dimension by degrading performance along other dimensions, as long as the performance satisfies the desired thresholds.

For example, consider the application of a transformation to a task with two performance objectives. Figure 3.5 shows the performance responses of the two performance dimensions d_1 and d_2 for the task and transformation. The dotted horizontal lines correspond to the two performance thresholds t_1 and t_2. Applying the transformation i times satisfies the performance objective on d_1 in Figure 3.5a, but not the objective on d_2 in Figure 3.5b. Likewise, applying the transformation j times satisfies the t_2 objective, but not the t_1 objective. Instead of settling for lower performance on both dimensions, if the performance threshold on dimension d_1

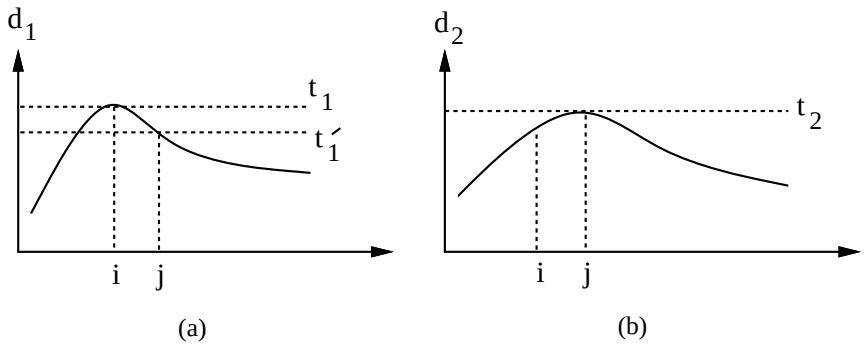

Figure 3.5: The performance responses for a task and transformation along two performance dimensions d_1 and d_2 illustrating the tradeoff between performance objectives. If performance dimension d_1 is content with threshold t_1' instead of t_1, then d_2's threshold is satisfiable at point j.

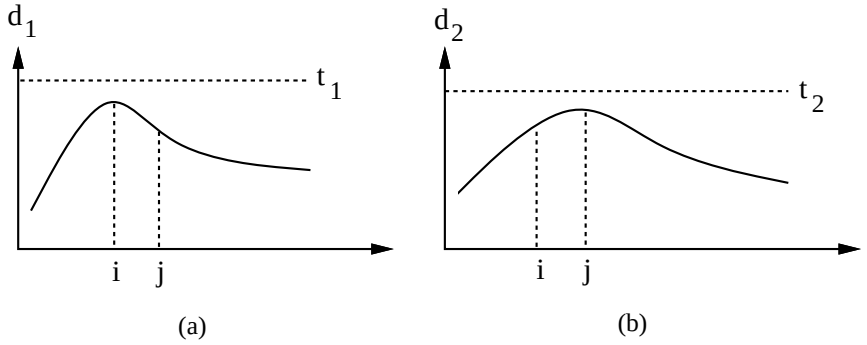

Figure 3.6: Unachievable performance objectives. Neither of the task's performance objectives (d_1, t_1) or $d_2, t_2)$ are achievable using the transformation.

were lowered to t_1', then both objectives are achievable by applying j transformations. This performance tradeoff is possible only with explicitly-defined performance objectives.

3.5.3 Unachievable Objectives

Performance objectives are not always achievable. For example, Figure 3.6a shows a case in which the model cannot achieve the desired performance threshold t_1 on the performance dimension d_1. If the threshold on d_1 is the only performance objective for the task, then the MBAC approach would recommend the maximum of the performance response (point i). Assuming the model is differentiable, this point can be determined analytically. Thus, MBAC settles on the closest point to the desired threshold.

If more than one performance objective is unachievable, the choice of the best number of transformations becomes more difficult. Consider the case where Figure 3.6b represents a second unachievable performance objective for the task. Attempting to maximize either performance response results in a degradation of performance for the non-maximized performance dimension.

The problem of determining the best compromise among several performance objectives is known as *multiobjective optimization* [Rao, 1984]. Methods for solving multiobjective optimization problems generally follow one of two approaches. One approach combines the individual objectives into a global objective, and then minimizes (or maximizes) the global objective. If the objectives are without thresholds, a weighted sum of the individual objectives is a good candidate for the global objective. If, as in the case of MBAC, the objectives have desired thresholds, then a least-squares regression of the objectives from the thresholds can serve as the global objective to be minimized. The second approach orders the individual objectives by importance, maximizing less important objectives according to constraints generated while maximizing the more important objectives. Assuming the user can order the objectives, this approach is analogous to the technique of lowering certain performance thresholds as described in the previous section.

3.5.4 Stopping Criterion

The stopping criterion of the MBAC algorithm in Figure 3.3 requires satisfaction of all performance objectives for the current task. However, the previous section shows that this criterion may be impossible. When satisfaction of all objectives is impossible, MBAC will either recommend making no transformations or oscillate between one or more transformations.

Figure 3.6a illustrates one case in which an objective is unsatisfied and MBAC suggests no transformation. After suggesting i transformations, the model can move performance no closer to the threshold, and MBAC suggests no transformations. The alternative case occurs when the performance threshold is below the performance achievable by the model; however, this case is unlikely to occur in practice. One method for handling this violation of the stopping criterion is to simply augment the criterion with "or the suggested transformation is null." However, if the behavior of the performance element is dynamic, retaining the original stopping criterion allows the models to continually update and adapt to changes in the performance element.

Figure 3.5 illustrates the case where MBAC may oscillate between transformations. Assuming the performance dimensions d_1 and d_2 have performance thresholds t_1 and t_2, respectively, performing i transformations achieves the d_1 objective, but violates the d_2 objective. If MBAC suggests the transformation anyway (i.e., the certainty of achieving the d_2 objective is high), then MBAC will then attempt to achieve the unsatisfied d_2 objective by making j transformations. Thus, MBAC oscillates between i and j transformations, never satisfying both objectives. If MBAC employs one of the multi-objective optimization techniques described in the previous section, then the models will compromise on an intermediate transformation, and the oscillation will degenerate into the null transformation case above. Without these techniques, MBAC may either employ loop detection to identify the oscillation or allow the oscillation to continue in the hope that performance element dynamics will eventually reveal another transformation for achieving all the objectives.

In the tradition of analog adaptive control systems, the simplest solution is to continue the control loop indefinitely. Enforcing a limit on the number of data points retained by each model prevents exhaustion of storage resources and ensures that the models and knowledge adapt to the more recent behavior of the performance element.

3.6 Model

One of the main assumptions of this thesis is the existence of a single model relating performance to the amount of learned knowledge. The model is applicable regardless of the task, performance dimension or knowledge transformation. Chapter 2 provides both experimental and formal support for this assumption. This section discusses the validity of the single-model assumption, the MBAC constraints on the model, alternative model types, model identification and transformation selection.

3.6.1 Validity

The proposed model expresses performance as a function of the amount of learned knowledge. The model's independence from the task and the performance dimension raises questions as to the validity of the model. First, the performance of a knowledge transformation depends on characteristics of the task domain (e.g., number of examples, number of features, size of

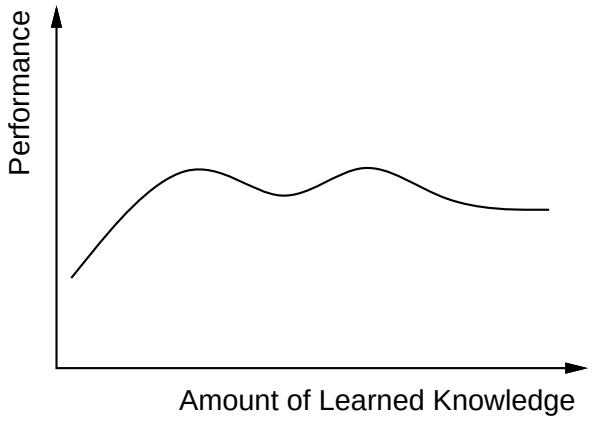

Figure 3.7: Performance response with multiple peaks.

instance space, etc.). How can a model, whose expression is independent of these characteristics, accurately predict performance? Second, the shape of the performance response depends on the method used to measure performance. How can a model based on one general shape accurately describe different performance measures?

The analysis of empirical learning in Section 2.5.1.1 answers the first question. This analysis derives constraints on the performance response based on characteristics of the domain: the number of training instances and the size of the instance space. The resulting performance response follows the empirically demonstrated trend of Figure 2.1. Although the values for the domain characteristics affect the shape of the performance response, the overall trend remains the same. Using a single parameterized curve as the model allows adaptation to these subtle effects and alleviates the need to derive similar constraints for other knowledge transformations.

As for the second question, there are some performance dimensions whose response does not match the trend. For example, regardless of the number of cycles performed on a neural network, the CPU time expended while using the resulting network to classify a set of test examples remains the same (assuming the topography of the network remains constant). However, neural networks are designed to improve accuracy, not CPU time. For the performance response to follow the general trend, the performance measure must depend on the performance affected by the transformation. For example, empirical learning methods generally attempt to improve solution accuracy; whereas, analytical learning methods attempt to improve solution time.

A more difficult issue underlying the second question is the possibility that the performance response has multiple peaks, as in Figure 3.7. If the knowledge transformation follows the

general to specific order described in Sections 2.5 and 3.4.2, this scenario is unlikely to occur. However, although a second-order model would be unable to precisely fit this fourth-order behavior, the second-order model can fit one of the peaks. If the MBAC approach constrains the model to fit the highest peak, then the resulting curve will ignore the local maxima associated with other peaks.

3.6.2 Constraints

The MBAC approach constrains the model to have four properties. First, the model must be adaptive. MBAC maintains several models, one for each combination of task, performance dimension and knowledge transformation. As discussed in the previous section, differing characteristics of the three components of the model cause the performance response curve to deviate slightly from the specific response curve of Figure 2.1. The MBAC approach identifies these deviations through experience by observing actual performance values for different amounts of learned knowledge. Therefore, the model must adapt to the experience. In addition to handling deviations from a specific response curve, an adaptive model adjusts to changes in other components of the MBAC approach. For example, the characteristics of the performance environment may change due to other loads on the computer system, installation on a different system, or installation of a different version of the performance element. Over time, the MBAC approach adapts the models accordingly, which in turn transforms the knowledge so as to maintain the performance objectives in the new performance environment.

Second, the model must be efficient to compute. Observation of new data points relating performance to amount of learned knowledge requires recomputation of the model. Since the MBAC approach does not incorporate the cost of model computation, the complexity of the computation should be no more than polynomial in the number of data points.

Third, the model must be accurate. Model accuracy represents a tradeoff between fitting deviations of the response curve and maintaining the desired trend of Figure 2.1. Therefore, MBAC requires the model to conform closely to the trend. The parameters of the model permit deviations without sacrificing this conformity.

Finally, MBAC must be able to measure the certainty of the model. In order for MBAC to trade off alternative knowledge transformations, the model must measure the certainty with which the suggested control decision will achieve the desired performance objectives. The cer-

tainty of the model derives from the deviations between the model and the actual performance response. One expression for the error depends on the standard deviation of the performance values at each point along the amount of learned knowledge.

3.6.3 Model Types

Using the MBAC model constraints of the previous section, this section evaluates alternative model types. The alternatives vary along the dimensions of precision of fit to the data and conformity to the general utility problem trend of Figure 2.1. The adaptive model types utilize performance curves observed over several trials.

The most precise model type with no concern for conformity to the trend is the *rote* model. The rote model retains each data point sampled from the actual performance response curve. When asked how many transformations are necessary to achieve a certain threshold of performance, the rote model finds the point whose performance value is closest to the desired threshold and returns the corresponding number of transformations. The rote model is adaptive and efficient, but not accurate in conforming to the trend. That is, the rote model overfits, because points from an anomalous response curve will hide the average response of the knowledge transformation. The rote model can estimate the certainty of achieving the performance threshold by computing the standard deviation of all performance values corresponding to the same number of transformations.

Several alternative model types ease the preciseness of the rote model. Instead of choosing a single point nearest the performance threshold, a nearest-neighbor model might choose several nearby points and average the suggested number of transformations. Further imprecision is possible by using an empirical learner similar to those in Chapter 2 to learn a discretized version of the performance response curve. As with the rote model, these model types are adaptive, efficient and can estimate certainty; however, they still have poor accuracy due to their neglect of the trend identified in Chapter 2.

At the other end of the spectrum lie model types more dependent on the trend, but less efficient to compute. One way to model the trend is as a sum of two curves similar to the approach taken in the formal analysis of Section 2.5. Figure 3.8 shows two curves whose mathematical expressions are

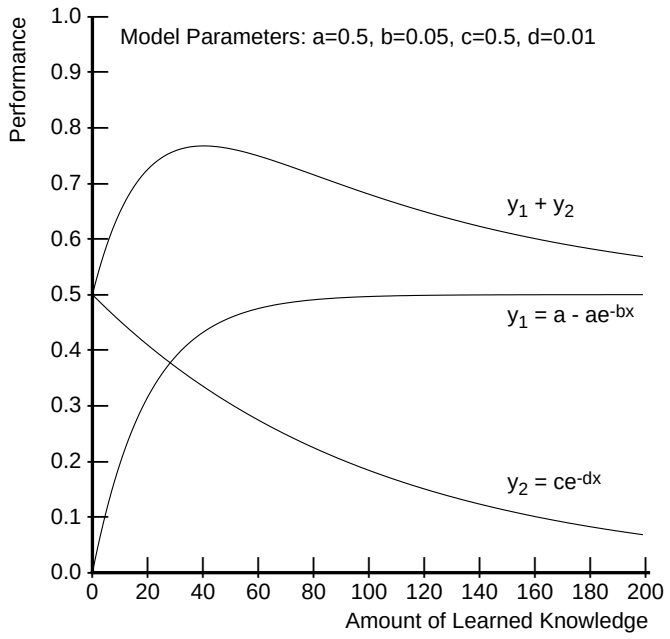

Figure 3.8: Nonlinear model of performance response trend as the sum of two component curves y_1 and y_2.

$$y_1 = a - ae^{-bx} \quad , \quad y_2 = be^{-dx}$$

where x varies with the amount of learned knowledge and a, b, c and d are model parameters. As Figure 3.8 illustrates, the sum of these curves $y_1 + y_2$ conforms to the trend. This model type is adaptive, accurate to the trend and conducive to certainty measure. However, since the model type is nonlinear in the parameters, fitting the model to the data is less computationally efficient than the previous methods due to the need for iterative minimization methods to find the model parameters [Press *et al.*, 1986].

The MBAC approach requires a model type which is a compromise between the two extremes represented by the above types of models. The model type must be efficient to compute and accurate to the trend. One such model type is the parabolic model shown in Figure 3.9. The parabolic model assumes that most performance objectives have thresholds near the peak of the performance response. In this case, modeling the peak is sufficient for controlling the amount of learned knowledge near the peak. The expression of the parabolic model is

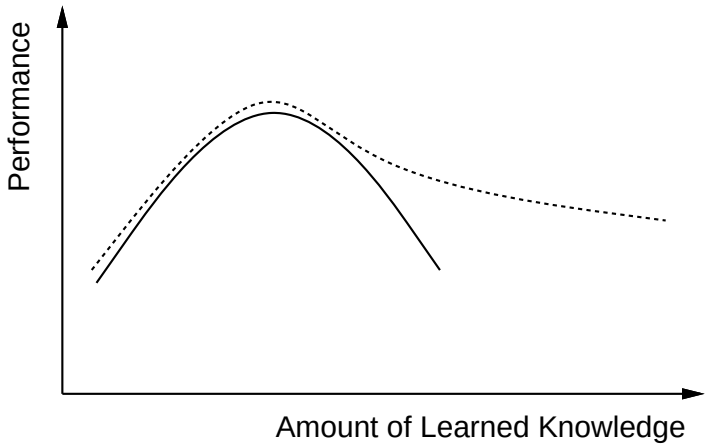

Figure 3.9: Parabolic model of performance response trend.

$$y = ax^2 + bx + c$$

where a, b and c are model parameters. The parabolic model is adaptive due to the variability of the parameters. The parabolic model is efficient to compute, because the expression is linear with respect to the model parameters. Assuming thresholds are near the peak, the parabolic model is accurate to the peak portion of the performance response trend. Press *et al.* [1986] discuss several methods for measuring the certainty of a linear model.

The parabolic model meets the constraints of the MBAC approach. Later discussions on implementation in Chapter 4 describe the exact methods for computing the parabolic model and measuring certainty. Experiments in Chapter 4 evaluate the utility of the parabolic model within the MBAC approach.

3.6.4 Model Identification

After the performance element attempts the task using the current knowledge, the MBAC algorithm of Figure 3.3 identifies the models pertaining to the task and the associated performance objectives. If the models predict performance on the task with high certainty (i.e., many data points support the model), then MBAC can use the techniques of Section 3.5 to make transformation decisions. However, some models may depend on few data points. Or, if the task is new, no data points exist from which to compute a model. This section considers the model identification process when the task models are weakly-supported or nonexistent.

3.6.4.1 Nonexistent Models

In the case where no models exist for a performance objective of the task, MBAC may either randomly select the type and amount of knowledge transformation or utilize other models to make a more informed decision. A random decision acquires a data point for the nonexistent model. The random decisions continue until MBAC has enough data points to compute the model (e.g., three points determine a parabola). Now that the model exists, MBAC makes decisions using the model to continue the collection of data points and increase model certainty.

If other models exist relating transformations to the performance objective, then MBAC can utilize these models to improve upon the random decision. One method for utilizing these models is to compare characteristics of the current task domain to other existing task domains. Example characteristics include the number of examples or the size of the instance space. The decisions made by models of similar tasks would be more applicable than a random decision. However, just as the model attempts to avoid using domain characteristics as parameters, the MBAC approach should avoid such characteristics for comparing models. Another method for utilizing other models assumes that a particular knowledge transformation has similar performance response curves for all tasks. Using this assumption, MBAC can average existing models to compute a decision for the current unknown task. Although this assumption is definitely not true in general, the averaged decision of existing models may be better than a random decision.

3.6.4.2 Weakly-Supported Models

Using the techniques of the previous section, MBAC gathers enough data points to compute the parameters of the model for the unknown task. However, the weakly-supported model may still perform poorly due to the lack of data. Although the techniques of the previous section still apply to the problem of collecting more data, the existence of a model permits another alternative. Instead of averaging all other models for the performance objective, MBAC may use the existing model to select only similar models, where similar means the difference between the curve of the new model and other model curves. The decisions of the more similar models have higher weight than less similar models in determining the transformation decision.

Another technique for utilizing other models along with a weakly-supported model is to compare the models' certainties. The weakly-supported model will have low certainty until the

collection of more data points. If a similar model exists with a higher certainty, MBAC can use the similar model's decision until the certainty of the new model rivals that of other models.

These methods for making decisions when tasks have nonexistent or weakly-supported models allow MBAC to make more informed transformation decisions in unknown task domains. If the user does not have time for MBAC to acquire performance models for a new task, then these methods are necessary. If the user permits MBAC to experiment with the task and acquire well-supported models, the resulting knowledge will more likely achieve the desired performance objectives.

3.6.5 Transformation Selection

After identifying appropriate models for the current task, MBAC uses the models to select a knowledge transformation for achieving the performance objectives of the task. The method for selecting a transformation depends on the number of performance objectives. If the task has only one performance objective, MBAC selects the transformation having low cost and high certainty of achieving the objective according to the models. Each model suggests some number of transformations for achieving the objective. MBAC uses the transformation cost to compute the total cost of performing the transformations. The predicted performance upon applying the transformation and the certainty of the model determine the certainty of achieving the objective. If the most certain model predicts achievement of the unsatisfied objective with zero transformations, then MBAC may choose the second highest certainty model for suggesting a transformation.

When a task has more than one performance objective, MBAC must consider the effect of transformations on each performance objective. If a transformation satisfies the unsatisfied performance objectives without violating the already satisfied objectives, MBAC selects the most certain of these transformations. As illustrated in Section 3.5.3, MBAC may be unable to satisfy all performance objectives for a task. Section 3.5.3 discussed approaches for selecting a transformation that minimizes the discrepancies between the objectives and achievable performance. When not all objectives are achievable, MBAC selects this "best compromise" transformation.

The next chapter describes the implementation of MBAC's transformation selection procedure, as well as other components of the MBAC approach. Experimentation using these

implementations evaluates the ability of the MBAC approach to maintain the utility of learned knowledge.

Chapter 4

Implementation and Evaluation

This chapter evaluates components of the MBAC approach discussed in Chapter 3. The first section describes the implementation of the knowledge transformations. The remaining sections describe more specific implementation details and present experiments that evaluate the MBAC approach. Experiment 1 (Section 4.2) evaluates MBAC's ability to converge to the peak of the performance response using different model types. Experiment 2 (Section 4.3) evaluates the estimation of model certainty. Experiment 3 (Section 4.4) evaluates MBAC's ability to select from among several knowledge transformations. Experiment 4 (Section 4.5) illustrates MBAC's dynamic behavior during initial adaptation of the performance response model. Experiment 5 (Section 4.6) evaluates MBAC's ability to transfer knowledge from known tasks to unknown tasks. Section 4.7 summarizes the experimental results.

4.1 Knowledge Transformations

Section 3.4 discussed four knowledge transformation issues: granularity, order, reversibility and cost. Sections 4.1.1 – 4.1.4 describe the implementation of the first three issues for the knowledge transformations used to evaluate the MBAC approach. The transformation cost is not implemented. Table 4.1 summarizes the implementation of the knowledge transformations.

4.1.1 ID3

Section 2.2.1.1 describes the ID3 empirical learning method (see also [Quinlan, 1986]). MBAC maintains knowledge for the ID3 transformation in the form of an n-ary decision tree. Although

most implementations of ID3 perform pre-pruning or post-pruning, Section 2.2.1.1 illustrates that ID3 with pruning still exhibits the utility problem and may decrease the peak of the performance response. The MBAC version ID3 transformation performs no implicit pre-pruning or post-pruning in order to increase the possibility of observing the maximum peak of the performance response. The granularity of the ID3 transformation is one n-ary split that generates n children from a node in the tree according to the n values of the split feature for the node. MBAC uses a breadth-first traversal of the decision tree for determining where to make the next split. The performance response curves in Section 2.2.1.1 indicate that the breadth-first order and the n-ary split granularity are sufficient to perceive the performance response trend. The reverse transformation removes the most recent split according to the breadth-first order.

4.1.2 PLS1

Section 2.2.1.2 describes the PLS1 empirical learner (see also [Rendell, 1983]). MBAC maintains knowledge for the PLS1 transformation in the form of a binary decision tree. The PLS1 transformation sets $t_\alpha = 1.0$ in the dissimilarity measure (see Section 2.2.1.2). A lower value for t_α allows PLS1 to make more splits. Since recommended values range from 1.0 to 2.0, this value for t_α is the lowest in the recommended range, and therefore allows the most splitting and the least chance of not reaching the peak of the performance response. The granularity of the transformation is one binary split generated by selecting a feature/value pair on which to split the region represented by the node of the decision tree. MBAC uses a breadth-first traversal of the decision tree for determining where to make the next split. The performance response curves in Section 2.2.1.2 indicate that the breadth-first order and the binary-split granularity are sufficient to perceive the performance response trend. The reverse transformation removes the most recent split according to the breadth-first order.

4.1.3 BackProp

Section 2.2.3 describes the error back-propagation (BackProp) neural network empirical learner (see also [Rumelhart *et al.*, 1986]). MBAC maintains knowledge for the BackProp transformation in the form of a network of nodes and weighted links. Although other network topologies exist, MBAC considers networks containing only three fully-interconnected layers: input layer, hidden layer and output layer. The task determines the number of nodes at the input and

output layers by assigning one input node to each feature/value pair and one output node to each class. The experiments in later sections refer to the BackProp method with n hidden units as BPn.

The granularity of the transformation is five cycles, where a cycle consists of one pass through the training instances with weight changes following each incorrectly classified instance. MBAC executes cycles in sequential order beginning with randomly-assigned weights. The performance response curves in Section 2.2.3 indicate that the sequential order and the five-cycle granularity are sufficient to perceive the performance response trend. The reverse transformation records the state of the network after every five cycles. When the model suggests a reverse transformation that is not a multiple of five, MBAC moves to the nearest recorded network.

4.1.4 Planner

Section 2.3.4 describes the Planner analytical learner. MBAC maintains knowledge for the Planner transformation as an ordered set of task-specific operators. Initially, this set contains the original operators for the task. MBAC inserts a learned operator (macro) into the set just before the original operators, but after previously-learned macros. This ordering has proved beneficial in other analytical learning work [Shavlik, 1988]. The granularity of the transformation is the addition of one macro. MBAC acquires this macro by selecting the next planning problem in an ordered list of randomly-selected problems from the task domain and generating a macro from the solution to this problem.[1]

The performance response curves in Section 2.3.4 indicate that the order and granularity are sufficient to perceive the performance response trend. The reverse transformation removes the most recently generated macro and repositions the pointer into the list of training problems to the example that generated this macro.

Table 4.1 summarizes the implementation of the knowledge transformations. Although the Planner transformation is available to MBAC, experimental results for this transformation are not included due to the lack of other analytical learners for comparison. The remainder of this chapter uses the empirical learning transformations to evaluate the MBAC approach.

[1] If a previously-learned macro completely solves the problem, MBAC selects the next problem in the ordered list of planning problems. This process continues until the generation of a new macro.

Table 4.1: Implementation of knowledge transformations.

Transformation	Knowledge	Granularity	Order	Reversibility
ID3	n-ary tree	n-ary split	breadth-first	n-ary unsplit
PLS1	binary tree	binary split	breadth-first	binary unsplit
BackProp	network	five cycles	sequential	five uncycles
Planner	operators	learn macro	exemplar	unlearn macro

4.2 Experiment 1: Convergence to Peak

The first experiment evaluates MBAC's ability to converge to the peak of the performance response using three different model types: rote, nearest-neighbor, and parabolic. The results of the comparison confirm the recommendation of Section 3.6 for the parabolic model. The next section describes the implementation of these model types. Section 4.2.2 discusses the method and results of the experiment.

4.2.1 Model Implementations

Section 3.6 introduces several candidate model types for the performance response curve. This section describes the implementation of the three model types (rote, nearest-neighbor, parabolic) used in Experiment 1. Given a model type, MBAC maintains a separate instance of the model type for each combination of task, knowledge transformation, and performance dimension. Given a threshold on the performance dimension of a task, the model estimates the number of transformations necessary to achieve (or come closest to) the threshold. The model predictions derive from the performance response curves sampled during the execution of MBAC.

4.2.1.1 Rote and Nearest-Neighbor Models

Given a performance threshold, the rote model estimates the number of transformations for achieving the threshold as the number of transformations corresponding to the sampled data point whose performance value is closest to the threshold. In case of ties, the rote model's estimate is the data point with the greater number of transformations.

The nearest-neighbor model is a generalization of the rote model. Instead of the closest data point to the threshold, the nearest-neighbor model estimates the number of transformations for achieving the threshold as the average number of transformations corresponding to the k closest data points to the threshold.. Experiment 1 arbitrarily sets $k = 5$.

4.2.1.2 Parabolic Model

MBAC computes the parabolic model by fitting a parabola to the data points sampled from the performance response curve. The performance response sample points in Step 1 of Figure 4.1 illustrate a possible initial state of the parabolic model. Due to previous experience, MBAC has acquired several data points measuring performance at different numbers of knowledge transformations. Since the points at higher numbers of transformations deviate from the parabolic model, MBAC must ignore these points when computing the model parameters. Several statistical methods exist for dealing with such outlier points [Press *et al.*, 1986]. MBAC uses a simple method (described below) for ignoring outlier points. Assuming performance thresholds near the peak of the performance response, the MBAC adaptive algorithm (see Section 4.5) rarely investigates points far beyond the observed peak.

Computation of the model parameters proceeds in three steps as shown in Figure 4.1. The first step finds the transformation number n corresponding to the data points having the highest average performance (the arrow in Step 1 of Figure 4.1). Step 2 removes the data points whose number of transformations is greater than $n + 1$. There are two reasons for choosing $n + 1$ as the cutoff. First, if there are no data points beyond n, MBAC may predict the peak to be at a higher number of transformations than n. Second, if there are points beyond n, the $n + 1$ point identifies the turning point of the parabola; whereas the $n + i$ $(i > 1)$ points do not lie on the portion of the performance response estimated by the parabolic model. Step 3 fits a parabolic curve to the resulting set of data points. Given the instantiated parabolic model, MBAC can analytically determine the number of transformations necessary to reach a point (performance threshold) on the curve, or the peak of the curve when the threshold is unachievable.

MBAC employs the chi-squared curve-fitting method [Press *et al.*, 1986] for computing the parameters of the parabolic model. Given a set of N data points (x_i, y_i, σ_i) (σ_i is the standard deviation in the i^{th} data point) and a parameterized model $\hat{y}(x_i)$, the chi-squared fitting method computes values for the model parameters minimizing the chi-square measure:

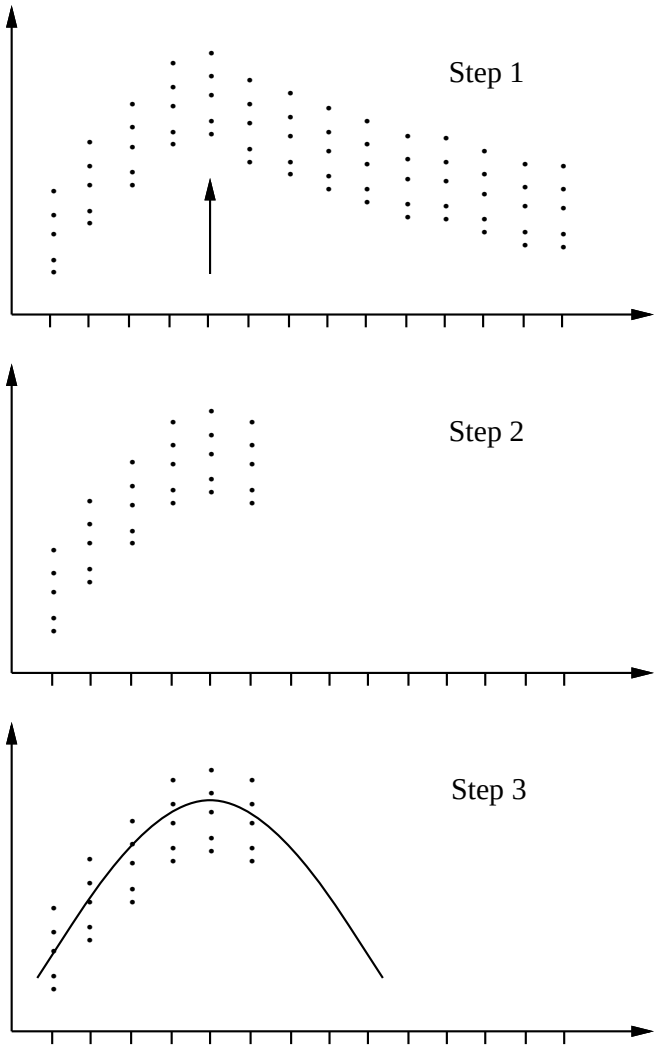

Figure 4.1: MBAC's three step process for parabolic curve fitting.

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \hat{y}(x_i)}{\sigma_i} \right]^2$$

MBAC obtains the σ_i's by computing the standard deviation in y_i (performance) for points having the same value for x_i (number of knowledge transformations). For the parabolic model, $\hat{y}(x_i) = ax_i^2 + bx_i + c$, where a, b and c are the model parameters. The values for a, b and c minimizing χ^2 occur at the point where the partial derivatives of χ^2 with respect to the parameters equal zero:

$$\frac{\partial \chi^2}{\partial a} = \frac{\partial \chi^2}{\partial b} = \frac{\partial \chi^2}{\partial c} = 0$$

Computing the derivatives leads to the following system of equations:

$$\begin{bmatrix} S_{x^2} & S_x & S \\ S_{x^3} & S_{x^2} & S_x \\ S_{x^4} & S_{x^3} & S_{x^2} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} S_y \\ S_{xy} \\ S_{x^2y} \end{bmatrix}$$

where

$$S_{x^j y^k} = \sum_{i=1}^{N} \frac{x_i^j y_i^k}{\sigma_i^2}.$$

Solving the system of equations using standard matrix techniques (e.g., Cramer's rule) yields the parameters for the parabola which most closely fit the data according to the χ^2 measure.

Figure 4.2 details the Estimate-Parabola procedure. First, the procedure collects all the unique x values in the given set of points. Estimate-Parabola then computes the mean and standard deviation of the set of y values at each x value. The procedure determines the x value *x-max* corresponding to the maximum average y value and builds a new set of points, where each point now contains the standard deviation σ_i. From *x-max*, the procedure determines the next greater value for x (which may not be *x-max + 1*) and filters out all points whose x value is beyond this value. Finally, the procedure estimates the parabola using the chi-squared curve fitting technique described above on the filtered points and returns the parabola. The parabola consists of the values for the three parameters a, b and c. However, if the chi-squared method cannot compute a parabola (e.g., less than three unique x values in the set of filtered points), then Estimate-Parabola returns *nil*.

```
procedure Estimate-Parabola (points)
begin
  x-max = 0
  y-max = 0
  new-points = {}
  x-values = unique-x-values(points)
  foreach x in x-values do
    y-values = collect-points-with-x-value(x, points)
    y-avg = mean(y-values)
    y-sigma = standard-deviation(y-values)
    if y-sigma = 0 then y-sigma = 1
    if y-avg > y-max then x-max = x, y-max = y-avg
    foreach y in y-values do
      new-points = new-points + {(x, y, y-sigma)}
  x-max-plus-1 = next-greater-x-value(x-max, x-values)
  new-points = test(new-points, x ≤ x-max-plus-1)
  parabola = chi-squared-fit(new-points)
  return(parabola)
end
```

Figure 4.2: Procedure for estimating a parabola from a set of points.

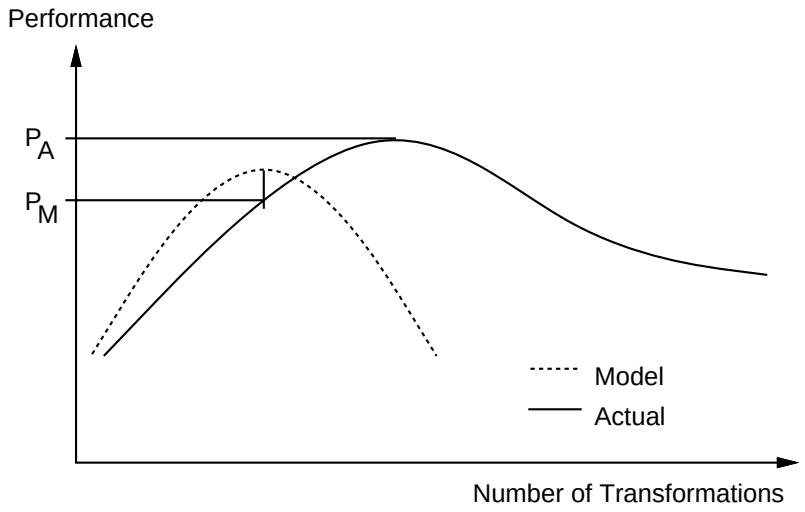

Figure 4.3: Experimental method for Experiment 1. P_A is the actual peak along the response curve, and P_M is the performance along the response curve corresponding to the peak of the model.

4.2.2 Method and Results

Avoidance of the general utility problem requires that the amount of learned knowledge correspond to the peak of the performance response. Experiment 1 evaluates MBAC's ability to converge to this peak. The experiment follows the same method for each knowledge transformation. Figure 4.3 graphically illustrates the experimental method. First, the method generates ten performance response curves. Each response curve is the result of learning on a randomly-selected set of training examples and testing on a randomly-selected set of test examples. Given the data points from the ten curves, MBAC computes the rote, nearest-neighbor and parabolic models and determines the number of transformations needed to reach the peak of the instantiated models. The dotted curve in Figure 4.3 shows this result for the parabolic model.

Next, the experimental method generates ten testing response curves using the same technique described above on another ten randomly-selected training and testing sets. The method finds the actual peak P_A of the average of the ten response curves. For each model type, the method measures the performance P_M along the testing response curve corresponding to the number of transformations suggested by the model. For the parabolic model example in Figure 4.3, P_M is the performance along the testing response corresponding to the peak of the instantiated parabola. Table 4.2 lists the average percentage of P_M/P_A for the empirical learning transformations[2]. Values are tabulated for the rote, nearest neighbor, and parabolic models over several task domains. The bottom row of Table 4.2 shows the average percentage over all transformations for each task domain.

The average percentage of the three model types over all task domains and all transformations in Table 4.2 is 94.2 for rote, 95.3 for nearest-neighbor, and 97.2 for parabolic. Thus, the parabolic model performs best of the three models at converging to the peak of the actual performance response. The nearest-neighbor model performs best on the Flag and Voting tasks, because the underlying concepts are less complex than the other tasks in terms of the number of transformations in the complete response curves. The fewer number of transformations allows the nearest-neighbor parameter setting ($k = 5$) to capture the average peak; whereas, the fewer number of transformations increases the difficulty of finding the correct parabolic model. Other values of k may not perform as well, and a method for choosing a proper k value is

[2] BPn in the table stands for BackProp with n hidden units

Table 4.2: Percentage measured peak performance of actual peak performance for empirical learners. Prediction from model type (rote, nearest-neighbor or parabolic) determines number of transformations at which to measure the peak.

	Breast Cancer			Flag			Flare			Voting		
	rote	near	para	rote	near	para	rote	near	para	rote	near	para
ID3	92.6	94.1	100	100	100	99.6	99.8	99.8	99.5	98.2	99.1	98.1
PLS1	100	99.1	98.4	99.8	99.8	99.8	99.9	99.0	99.6	100	99.6	99.7
BP2	91.3	90.9	94.7	89.8	96.0	87.0	79.1	77.2	99.6	99.7	99.5	99.1
BP4	94.8	96.5	95.9	96.1	95.5	93.9	89.7	89.7	95.8	99.5	99.8	99.5
BP8	97.5	97.9	99.4	94.1	94.3	92.4	93.1	92.4	93.3	99.9	99.9	99.9
BP16	80.3	82.6	96.2	96.4	96.4	99.0	70.4	88.4	94.1	99.5	99.5	99.5
Avg	92.7	93.5	97.4	96.0	97.0	95.3	88.7	91.1	97.0	99.5	99.6	99.3

unclear. Furthermore, the anomaly described below provides another explanation for why the nearest-neighbor model performs best on the Flag task. As noted in Section 3.6, the downfall of the rote and nearest-neighbor models is their tendency to overfit the performance response curve without the guidance of the known performance response trend.

One anomaly occurs in Table 4.2 for BP2 on the Flag domain. The values for this entry indicate that the nearest-neighbor model performs unusually better than the rote and parabolic models. Figure 4.4 elucidates the reason for this anomaly. The figure shows the average of the ten responses used to compute the models (jagged solid line), the computed parabolic model (smooth solid line), and the average of the ten testing responses (dotted line). Also shown are the number of transformations recommended by the rote, nearest-neighbor and parabolic model types (vertical solid lines). Due to the low number of hidden units and the initially random set of weights in the network, the back-propagation algorithm may exhibit erratic behavior as evidenced by the discrepancy between the average of the training and testing responses in Figure 4.4. In this specific instance, none of the models predict the true peak, and the prediction of the nearest-neighbor model happens to fall at a point much higher than the predictions of the other model types. This situation indicates the need for more training to average out the effects of such anomalies.

The existence of this anomalous behavior indicates the importance of data sampling issues for this and future experiments. Normally, a learning method generates knowledge based on a set of training examples and then evaluates the knowledge using a separate set of testing

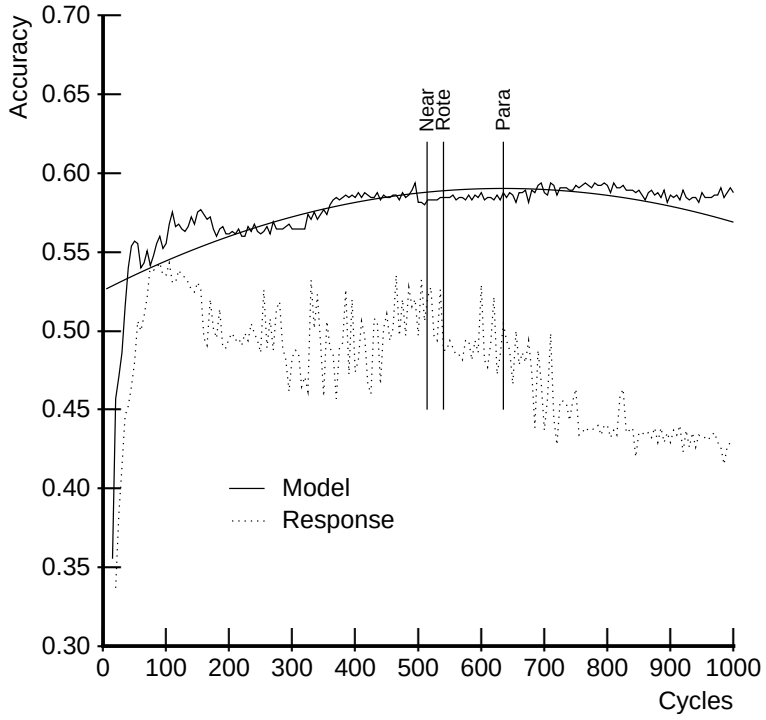

Figure 4.4: Anomalous behavior of BP2 on the Flag domain. The jagged solid line is the average of the ten responses used to train the model. The smooth solid line is the instantiated parabolic model. The dotted line is the average of the ten responses used for testing. The vertical solid lines show the number of transformations recommended by the three model types.

examples drawn from the same distribution as the training examples. The evaluation is testing the hypothesis that the learning method generalizes appropriately. However, the MBAC experiments are testing the hypothesis that a model type can capture the trend in the performance response of the learning method. Training and testing data sets draw from the set of possible performance responses. One performance response from this set is the result of measuring the performance of a learning method, while learning on a set of training examples, on a set of testing examples. Therefore, the data set from which these experiments are sampling has an element for each combination of training and testing sets drawn from the original set of examples available with the task domain. The MBAC experiments randomly select training and testing response curves from this set of performance responses. The data sampling technique of using ten training and ten testing responses is a small sample of the set of possible responses, but each response is the result of running an entire learning method. This process is computationally expensive, especially for the back-propagation learning method. One future direction for the improvement of the experimental evaluation is the use of a cross-validation technique to improve the validity of the experimental results for the small number of samples.

Another perspective on the data from Table 4.2 is the difference between the actual peak performance P_A and the performance achieved using the model's prediction P_M. Comparison of this difference with the standard deviation of the parabolic model indicates whether the error in $P_A - P_M$ is within the error (one standard deviation) of the model. The standard deviation of the parabolic model is the average absolute difference between each point used to fit the model (Step 3 of Figure 4.1) and the estimated parabola. At this point, we adopt the parabola as the model for the MBAC approach. Table 4.3 lists the average difference $P_A - P_M$ for the empirical learning transformations and indicates the standard deviation of the model in parentheses.

Table 4.3 indicates that the difference between actual peak performance and MBAC peak performance is within one standard deviation of the model. Thus, the parabolic model's predicted peak in the performance response is correct within the error of the model. These results further indicate the applicability of the standard deviation as a model certainty estimate. The next experiment evaluates the standard deviation and other measures as estimates of model certainty.

Table 4.3: Actual peak performance minus MBAC peak performance for empirical learners using the parabolic model. The standard deviation of the parabolic model appears in parentheses. The values are in units of classification accuracy.

	Breast Cancer	Flag	Flare	Voting
ID3	0.000(0.042)	0.003(0.063)	0.004(0.017)	0.019(0.076)
PLS1	0.012(0.037)	0.002(0.041)	0.003(0.017)	0.003(0.058)
BP2	0.015(0.117)	0.071(0.107)	0.002(0.257)	0.008(0.032)
BP4	0.020(0.138)	0.031(0.085)	0.028(0.189)	0.004(0.028)
BP8	0.003(0.077)	0.043(0.096)	0.027(0.163)	0.001(0.035)
BP16	0.018(0.112)	0.006(0.093)	0.028(0.148)	0.004(0.020)

4.3 Experiment 2: Model Certainty Estimation

The certainty of a model indicates the likelihood that the transformations suggested by the model will actually achieve their predicted performance. MBAC estimates the certainty of the parabolic model in order to select a promising transformation from among those suggested by different models. The next section describes three certainty estimators available to MBAC. Experiment 2 compares the different estimators for their ability to identify the best transformation. Section 4.3.2 discusses the method and results of Experiment 2.

4.3.1 Certainty Estimators

Experiment 2 evaluates three certainty estimators: standard deviation, normalized standard deviation, and model probability. This section describes the implementation of these certainty estimators.

The first certainty estimation method utilizes the standard deviation of the model's data points from the computed parabolic model. The standard deviation of the parabolic model is the average absolute difference between each point used to fit the parabola and the instantiated parabola. Figure 4.5 shows the parabolas at one standard deviation from the parabolic model of Figure 4.1. The standard deviation certainty measure is the same standard deviation measurement used in Experiment 1 (see Table 4.3). The lower the standard deviation (SD) in the model, the more certainty MBAC places in the model.

One problem with the standard deviation certainty estimator is the dependency on the scale of the performance response. The standard deviation of a response varying over a small

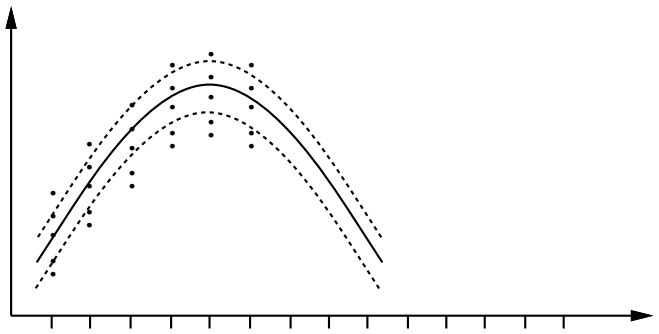

Figure 4.5: Bounding parabolas at one standard deviation.

range of performance values may appear tighter than the deviation for a response over a larger range of performance values, even though the latter model may be a better fit. The second certainty estimator attempts to overcome this problem by normalizing the standard deviation. Normalization expands the performance response so that the minimum value is at zero, and the maximum value is at one. Multiplying the standard deviation by the scale factor necessary to accomplish the normalization yields the normalized deviation. If the minimum and maximum values of the performance response are p_0 and p_1 respectively, then the normalized standard deviation (SDN) is

$$\text{SDN} = \frac{\text{SD}}{p_1 - p_0}$$

where SD is the standard deviation as described above. The lower the normalized standard deviation, the more certainty MBAC places in the model.

The third certainty estimator uses the value of the χ^2 measure from Section 4.2.1.2 to compute the probability Q that discrepancies from the model are due to chance, i.e., the model is a good fit to the data. Press *et al.* [1986] describe the method for computing Q from the value of χ^2 and the number of degrees of freedom $N - M$, where N is the number of data points and M is the number of model parameters ($M = 3$ for the parabolic model). The higher the probability Q, the more certainty MBAC places in the model.

4.3.2 Method and Results

Before choosing an appropriate transformation, MBAC must be reasonably certain that the parabolic model's predicted performance achievement will actually occur on unseen examples.

Table 4.4: Parabolic model certainty estimators, standard deviation SD, normalized standard deviation SDN, and model probability Q, for empirical learning methods over several task domains. The BEST column orders the transformations according to the performance achieved using the recommended number of transformations (1 = highest performance).

	Breast Cancer				Flag			
	BEST	SD	SDN	Q	BEST	SD	SDN	Q
ID3	2	0.042	0.198	0.359	2	0.063	0.254	0.314
PLS1	1	0.037	0.251	0.305	1	0.041	0.168	0.266
BP2	6	0.117	0.213	0.036	6	0.107	0.144	0.165
BP4	4	0.138	0.233	0.193	5	0.085	0.131	0.014
BP8	3	0.077	0.132	9.04e-7	4	0.096	0.136	0.060
BP16	5	0.112	0.183	0.417	3	0.093	0.151	0.055

	Flare				Voting			
	BEST	SD	SDN	Q	BEST	SD	SDN	Q
ID3	1	0.017	0.192	0.316	2	0.076	0.180	5.50e-132
PLS1	2	0.017	0.244	0.345	1	0.058	0.135	7.02e-67
BP2	4	0.257	0.394	0.470	6	0.032	0.090	0.027
BP4	5	0.189	0.290	0.331	4	0.028	0.113	0.021
BP8	6	0.163	0.247	0.448	3	0.035	0.138	0.156
BP16	3	0.148	0.235	0.362	5	0.020	0.110	0.039

Experiment 2 compares three different certainty estimators: standard deviation (SD), normalized standard deviation (SDN), and model probability (Q). The experiment proceeds by generating ten response curves for several task/transformation combinations. These curves serve as data for fitting the parabolic model, as in Experiment 1, and computing the three certainty estimators. Table 4.4 shows the three certainty estimates for the models of the empirical learning methods over several task domains.

Next, the experiment orders the transformations according to the performance achieved by executing the number of transformations recommended by the models. MBAC performs the transformations on a separate response curve that is an average over ten response curves generated from randomly-selected training and testing example sets for each task domain and transformation. Table 4.4 contains the ordering information in the BEST column.

One way to evaluate a certainty estimator is to observe the correlation between the estimator and the BEST ordering on the transformations in Table 4.4. The model probability certainty estimator Q should decrease as the order increases (i.e., better models have higher Q's). However, the Q values do not correlate well with the BEST ordering. For example, in

all but the Voting task domain, Q increases from the first to the second transformation in the ordering. Also, in all but the Flag domain, the highest Q value accompanies a transformation ordered three or higher. One reason for the poor correlation of Q values to the quality of the model is that the computation of Q assumes the data measurement errors σ_i have a normal distribution. The distribution of the measurement errors is unknown, and discrepancies from the normal distribution degrade the correlation of Q with the best transformation order.

For the normalized standard deviation certainty estimator (SDN), the values should increase as the transformation order increases (i.e., better models have lower deviations). However, the SDN values also do not correlate well with the BEST ordering. For example, in all but the Flare task domain, the lowest SDN value accompanies a transformation ordered three or higher. One reason for the poor correlation of the SDN values is the fluctuation at the minimum p_0 of the performance response. The minimum value of the performance response is usually the point at zero transformations and depends more on the variance within the training data than the standard deviation of the performance response. These fluctuations bias the normalization process away from the desired normalized standard deviation.

Of the three certainty estimators, the standard deviation (SD) has the best correlation with the transformation ordering. The SD values do not decrease from the first to the second transformation in the ordering, and in all but the Voting task domain, the lowest SD for each task domain accompanies the first transformation in the ordering. Furthermore, Table 4.3 of Experiment 1 shows that the difference between the actual and predicted peak of the performance response curve is within the standard deviation of the model, indicating that the SD values are not too low. Due to the superiority of the standard deviation certainty estimator, MBAC adopts SD as the model certainty estimator. Model certainty aids MBAC when deciding between transformations with similar predicted performance achievement.

4.4 Experiment 3: Transformation Selection

MBAC selects transformations according to the corresponding model's ability to achieve the performance threshold. Experiment 3 compares the parabolic model's predicted performance to the actual performance obtained by performing the recommended number of transformations. The next section describes the transformation selection procedure, and Section 4.4.2 presents

```
structure Model
begin
   task           ;task domain
   dimension      ;performance dimension
   transformation ;method for increasing/decreasing amount of learned knowledge
   knowledge      ;pointer to transformed knowledge
   index          ;current amount of learned knowledge
   data           ;sampled points from performance response
   certainty      ;model certainty estimate
   error          ;current value of (threshold – model's predicted performance)
   move           ;recommended number of transformations from index to reach threshold
end
```

Figure 4.6: Structure definition for the parabolic model.

the experimental method and results. The results indicate that the selected transformation achieves predicted performance and out-performs the unselected transformations.

4.4.1 Transformation Selection

In order to describe the transformation selection process, this section first describes the structure used for the parabolic model. Figure 4.6 shows the model structure definition. MBAC defines a model for each combination of task, performance dimension, and transformation. The first three fields of the model structure record this information. MBAC maintains a separate knowledge structure (e.g., decision tree or neural network) for each combination of task and transformation. The *knowledge* field points to this knowledge structure. The *index* is the number of transformations (from zero) made on the knowledge. The model *data* contains the set of performance response samples observed over time. The *certainty* field contains the value of the model's certainty estimate. The *error* field is a placeholder for the distance between the model's current predicted performance and a given performance threshold. The *move* field is a placeholder for the recommended number of transformations from *index* for achieving a given threshold.

The transformation selection procedure implemented for Experiment 3 handles only one performance objective. Figure 4.7 describes the Select-Transformation procedure, which takes a performance threshold and a set of models pertaining to the performance dimension of the threshold. The procedure returns the best model corresponding to the best transformation

```
procedure Select-Transformation (models, threshold)
begin
  foreach model in models do
     decision = Parabolic-Decision(model, threshold)
     model-move(model) = decision-move(decision)
     model-error(model) = |threshold − decision-performance(decision)|
  models = sort(models, model-error, <, model-certainty, >)
  best-model = pop(models)
  return(best-model)
end

procedure Parabolic-Decision (model, threshold)
begin
  index = model-index(model)
  p = Estimate-Parabola(model-data(model)) ;see Figure 4.2
  if exists(p) and concave-down(p)
  then model-certainty(model) = estimate-certainty(model)
       x1, x2 = solve(p, threshold)
       if complex(x1, x2)
       then x = (x1 + x2) / 2
       else x = lowest-positive(x1, x2)
       return(x − index, p(x))
  else return(nil)
end
```

Figure 4.7: Transformation selection procedure.

for achieving the threshold. The *move* field of the returned model contains the recommended move according to the parabolic model. First, Select-Transformation computes the decision made by a parabolic estimate of the model data for each model (see discussion of the Parabolic-Decision procedure below). A decision consists of a move (number of transformations) along the amount of learned knowledge axis and the predicted performance after performing the move. The model error is the absolute value of the difference between the threshold and the predicted performance. Next, Select-Transformation sorts the selected models in ascending order of model error. Models with the same error are further sorted in descending order of model certainty (see Section 4.3.1 for a discussion of model certainty estimation methods). The best model is the first model in the set of sorted models.

The Parabolic-Decision procedure (also shown in Figure 4.7) returns a decision for moving the given model to the given threshold. First, Parabolic-Decision estimates a parabola from the model data (see Estimate-Parabola procedure in Figure 4.2). If the parabola does not exist or is not concave down, the Parabolic-Decision returns *nil*. Otherwise, the procedure estimates the model certainty according to the method outlined in Section 4.3.1. Then, Parabolic-Decision solves for the x values $x1$ and $x2$ (number of transformations) whose y values along the parabola equal the threshold. If $x1$ and $x2$ are complex numbers, then the threshold is above the peak of the parabola, and Parabolic-Decision sets x to the average of $x1$ and $x2$, which is the x value at the parabola's vertex (peak). If $x1$ and $x2$ are real numbers, then Parabolic-Decision sets x to the lower positive value between $x1$ and $x2$. The procedure computes the recommended move as the difference between x and the model's current index. Parabolic-Decision returns the move and the performance at x predicted by the parabola.

4.4.2 Method and Results

Experiment 3 evaluates how well the transformation selection procedure in Figure 4.7 orders models according to their ability to achieve a desired performance objective. The experimental method proceeds similarly to previous experiments. First, the learning methods use randomly-selected training and testing sets to produce ten response curves for each combination of task domain and transformation. These curves provide the data points for fitting the parabolic models. Next, the same process generates ten more response curves to be used for testing the recommendations of the models. The performance threshold is set at a point higher than

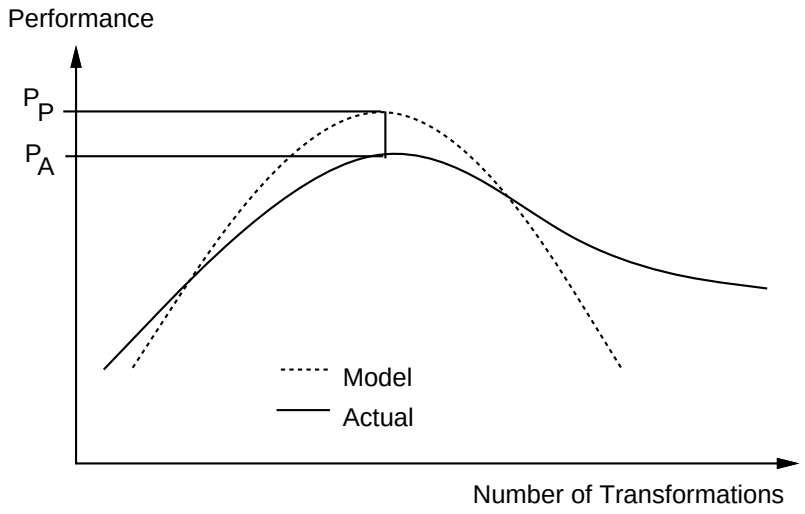

Figure 4.8: Experimental method for Experiment 3. P_P is the predicted peak according to the parabolic model. P_A is the actual performance along the response curve corresponding to the peak of the parabolic model.

that achievable by any learning method; therefore, the models predict their achievable peak performance. Figure 4.8 illustrates the experimental method. The model's predicted performance value is P_P, and the actual performance is P_A. Table 4.5 shows the model's predicted performance P_P, standard deviation, and the actual performance P_A achieved by performing the recommended number of transformations on the average of the ten testing response curves.

The results from Experiment 3 indicate that the model predictions are accurate. Since the transformation selection procedure in Figure 4.7 sorts the models based on their predictions, the procedure will perform well at choosing the best transformation. The results show that the actual performance achieved by the transformations is within one standard deviation of the predicted performance for all but BP2 and BP4 on the Flag task domain. Therefore, the model predictions are accurate to within one standard deviation. The results also show that the highest predicted performance correlates with the highest actual performance in all but the Flare domain (where the difference between the first and second best actual performance is small). Therefore, the transformation selection procedure, which picks the highest predicted performance, consistently chooses the best (or near best) transformation.

Along with previous experiments, Experiment 3 uses ten response curves for training and ten for testing. Another method for evaluating MBAC experimentally is cross-validation [Breiman

Table 4.5: Parabolic model's predicted performance and actual performance on empirical learning methods over four domains. The standard deviation of the model is shown in parentheses. Values are in units of classification accuracy.

	Breast Cancer		Flag	
	Predicted	Actual	Predicted	Actual
ID3	0.705(0.042)	0.697	0.762(0.063)	0.771
PLS1	0.712(0.037)	0.706	0.776(0.041)	0.795
BP2	0.353(0.117)	0.262	0.590(0.107)	0.472
BP4	0.403(0.138)	0.466	0.593(0.085)	0.477
BP8	0.501(0.077)	0.503	0.612(0.096)	0.526
BP16	0.450(0.112)	0.447	0.549(0.093)	0.591
	Flare		Voting	
	Predicted	Actual	Predicted	Actual
ID3	0.814(0.017)	0.815	0.959(0.076)	0.947
PLS1	0.825(0.017)	0.813	1.003(0.058)	0.960
BP2	0.393(0.257)	0.441	0.912(0.032)	0.903
BP4	0.407(0.189)	0.415	0.917(0.028)	0.911
BP8	0.404(0.163)	0.379	0.928(0.035)	0.927
BP16	0.412(0.148)	0.447	0.927(0.020)	0.908

et al., 1984]. The cross-validation method would start with, say, ten response curves. Then, leaving a different one of the ten out each time, the method uses the nine response curves for training, and the one for testing. The average of the results from ten such trials would be the entry in the experimental table. In this way, cross-validation improves the ability to evaluate a system with sparse data. The data for evaluating the MBAC approach is the set of possible response curves. Although the experimental method can generate an arbitrary number of response curves, each curve corresponds to executing an entire learning method, which may be computationally expensive. Cross-validation may offer an alternative to the computationally expensive generation of more response curves by making better use of a smaller number of curves.

At a lower level of detail, cross-validation may improve the performance of the individual learning methods. Instead of using two-thirds of the task domain data for training and one-third for testing (as described in Appendix A), one may split the data into n sets. Leaving a different set out each time, the method trains using $n-1$ of the sets, and tests the resulting hypothesis on the remaining set. The final hypothesis would be the one performing best over

the n trials, or an "average" of the best hypotheses. The use of cross-validation at this level constitutes another method that MBAC may select, but not a change to the MBAC control structure. MBAC may learn to select methods using cross-validation for task domains having few examples.

4.5 Experiment 4: MBAC Initial Dynamics

The previous three experiments show MBAC's behavior after considerable sampling of the performance response curve. Experiment 4 illustrates the initial dynamics of MBAC starting with no samples from the performance response. Results show that MBAC quickly acquires an accurate model of the performance response. The next section presents the MBAC adaptive control algorithm. Section 4.5.2 describes the method and results of the experiment.

4.5.1 Adaptive Control Algorithm

Figure 4.9 shows MBAC's adaptive control algorithm. This implementation handles only one performance objective (see Section 3.5 for a discussion of methods for handling multiple objectives). The MBAC procedure takes a task, a performance objective (consisting of a performance dimension and a threshold), and a set of available transformations. For each transformation, MBAC builds a model (see discussion of the Build-Model procedure below). Then, MBAC enters the main control loop which selects a model and performs the move recommended by the model. The actual number of transformations made by the Perform-Transformation procedure may be less than the recommended number. The loop continues until performance satisfies the objective or the actual number of moves made is zero.

The Build-Model procedure (also shown in Figure 4.9) returns a model structure containing the nine fields defined in Figure 4.6: task, dimension, transformation, knowledge, index, data, error, certainty and move. The *task*, *dimension* and *transformation* fields come directly from the inputs to the MBAC procedure. For example, a model might relate the accuracy dimension to the number of ID3 splits transformation for the Flag task. In this case, the knowledge is the decision tree. The initial model *knowledge* corresponds to the initial hypothesis of the transformation. For example, the initial hypothesis of ID3 is the majority class of the training examples of the task. The *index* field is the number of transformations (amount of learned

```
procedure MBAC (task, objective, transformations)
begin
  dimension = objective-dimension(objective)
  threshold = objective-threshold(objective)
  models = {}
  foreach transformation in transformations do
     models = models + Build-Model(task, dimension, transformation)
  repeat
    model = Select-Transformation(models, threshold) ;see Figure 4.10
    performance = Perform-Transformation(model) ;see Figure 4.11
  until (performance = threshold) or (model-move(model) = 0)
  return(model)
end

procedure Build-Model (task, dimension, transformation)
begin
  m = make-model() ;see Figure 4.6
  model-task(m) = task
  model-dimension(m) = dimension
  model-transformation(m) = transformation
  model-knowledge(m) = initial-hypothesis(task, transformation)
  model-index(m) = 0
  performance = evaluate(task, dimension, model-knowledge(m))
  model-data(m) = {(0, performance)}
  model-error(m) = nil
  model-certainty(m) = nil
  model-move(m) = nil
  return(m)
end
```

Figure 4.9: Model-based adaptive control procedure for one performance objective.

```
procedure Select-Transformation (models, threshold)
begin
  foreach model in models do
     decision = Parabolic-Decision(model, threshold) ;see Figure 4.7
     model-move(model) = decision-move(decision)
     model-error(model) = |threshold − decision-performance(decision)|
  best-models = test(models, |model-move| > 0)
  if best-models = nil then best-models = test(models, model-move=nil)
  if best-models = nil then best-models = models
  best-models = sort(best-models, model-error, <, model-certainty, >)
  best-model = pop(best-models)
  if model-move(best-model) = nil then model-move(best-model) = 1
  return(best-model)
end
```

Figure 4.10: Transformation selection procedure for adaptive MBAC.

knowledge) from zero currently represented by the knowledge for the transformation. For example, the index for the ID3 transformation reflects the number of splits in the current decision-tree representation of ID3's hypothesis for the task. The hypothesis at index=0 is the initial hypothesis. The *data* field contains the set of sampled points from the actual performance response. Initially, the data contains only one point: the performance of the knowledge at index=0. The performance value is measured by evaluating the task on the knowledge at index=0. Finally, Build-Model initializes the *error*, *certainty* and *move* to *nil* and returns the model.

The Select-Transformation procedure (see Figure 4.10) returns the model corresponding to the best transformation of those given to the MBAC procedure. This procedure is the same as the one described in Figure 4.7, but contains more detail not utilized in Experiment 3. The Select-Transformation procedure embodies MBAC's definition of the best transformation. First, Select-Transformation computes the decision made by a parabolic estimate of the model data for each model (see discussion of Parabola-Decision in Section 4.4.1). A decision consists of a move (number of transformations) along the amount of learned knowledge axis and the predicted performance after performing the move. The model error is the absolute value of the difference between the threshold and the predicted performance.

```
procedure Perform-Transformation (model)
begin
  actual-move = transform(model-knowledge(model), model-move(model))
  model-index(model) = model-index(model) + actual-move
  model-move(model) = actual-move
  performance = evaluate(model-task(model), model-dimension(model),
                         model-knowledge(model))
  model-data(model) = model-data(model) + {(model-index(model), performance)}
  return(performance)
end
```

Figure 4.11: Procedure for performing a transformation.

Next, Select-Transformation filters the models according to several tests. The first test selects models recommending non-*nil*, non-zero moves. If none exist, the second test selects models recommending *nil* moves. A *nil* move indicates that a parabolic model does not yet exist due to a lack of data points, and that more sampling is necessary. If no *nil* moves exist, then the procedure selects models recommending zero moves (the remaining models). Select-Transformation sorts the selected models in ascending order of model error and then in descending order of model certainty (see Section 4.3.1 for a discussion of model certainty estimation methods). The best model is the first model in the set of sorted models. If the best model's move is *nil*, the move is set to one to promote further sampling of the performance response.

After selecting the best transformation, MBAC performs the transformation by making *move* transformations on the corresponding model knowledge. The Perform-Transformation procedure shown in Figure 4.11 takes a model and performs the number of transformations stored in the *move* field of the model. The procedure begins by transforming the knowledge according to the move. The actual number of transformations made on the knowledge may be less than *move*. The procedure sets *actual-move* to the actual number of performed transformations. For example, the ID3 transformation transforms the knowledge by adding or removing splits according to a positive or negative move. If the current knowledge corresponds to 90 out of a possible 100 splits, and the move recommends 20 splits, then the actual move will be only 10 splits. After updating the model index, the procedure evaluates the new knowledge on the task using the performance element while monitoring the performance dimension.

Perform-Transformation updates the model data with the new performance point and returns the performance value.

4.5.2 Method and Results

Experiment 4 demonstrates the adaptive control algorithm of the previous section with three different transformations (ID3, PLS1 and BP16) on the Flag task. These transformations correspond to the three best transformations identified in Table 4.4. The experiment uses the Flag task, because the behavior is representative of the other tasks and simple enough to explain in detail. The experimental method defines the task by randomly selecting training and testing sets from the Flag examples as described in Appendix A. Next, the method calls the MBAC procedure with the task, the performance objective of classification accuracy=1.0, and one of the three transformations. The method retains the performance of the initial hypothesis and the performance after each iteration of the adaptive control loop. The experiment repeats this method nine times for each transformation.

Figure 4.12 shows an example to help explain the subsequent experimental results. The top graph plots the classification accuracy of the ID3 decision tree on the Flag task after each control iteration made by MBAC. The dashed line marks the peak performance of the performance response. This plot is hereafter referred to as a *control response*. A control response plots performance versus number of control iterations; whereas, the performance response plots performance versus number of transformations. The plots in Figure 4.12 correspond to the top-center control response in Figure 4.13. The bottom graph in Figure 4.12 plots the cumulative number of transformations (splits) made to the ID3 decision tree after each control iteration. The dashed line in this plot marks the number of transformations corresponding to the peak of the performance response. The vertical dotted line in both plots marks the control iteration at which the MBAC adaptive control procedure terminates.

At the beginning of the control response in Figure 4.12, before any control iterations, the accuracy is 0.72 with zero transformations (splits). Since the model has only one sample point $(0, 0.72)$, the Parabolic-Decision procedure returns *nil*, and the Select-Transformation procedure returns a move of one transformation. During the first control iteration, MBAC performs this transformation (one split), and accuracy improves to 0.80. With only two points, Parabolic-

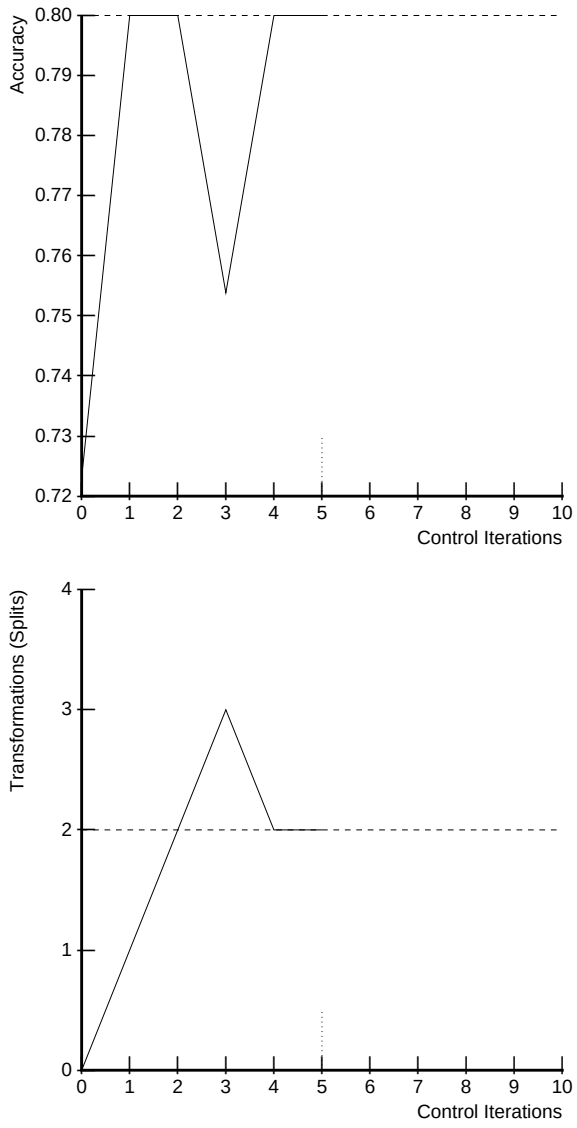

Figure 4.12: Explanation of experimental results for Experiment 4. The top graph plots accuracy versus control iterations for one instance of ID3 on the Flag task domain. The bottom graph plots transformations versus control iterations. The dashed lines mark the point of peak performance. The vertical dotted lines mark the last control iteration.

Decision again returns *nil*, and MBAC performs another single forward transformation (split). Accuracy remains at 0.8 after this second control iteration.

At this point, the model contains three data points, the minimum number needed to fit a parabola. However, MBAC has not yet observed a degradation in performance, and therefore does not know the location of the peak. When the last two points of the performance response are equal, the Estimate-Parabola procedure does not fit a parabola to the data in order to promote further investigation of the performance response. This situation did not arise in Experiment 1; therefore, the Estimate-Parabola procedure in Figure 4.2 does not describe this facet of the procedure. Thus, the Parabolic-Decision procedure again returns *nil*, and MBAC performs a third transformation (split). After this third control iteration, accuracy drops to 0.75.

With the degradation in performance, the Parabolic-Decision procedure is now able to identify the parameters of the parabolic model and suggests a single negative transformation (unsplit). MBAC performs the transformation during the fourth control iteration, and accuracy returns to 0.80. Since two transformations (splits) corresponds to the peak of the parabola, the Select-Transformation procedure recommends zero transformations (splits) for the fifth control iteration. Because the move is zero transformations, MBAC's adaptive control loop terminates after five iterations. Although each iteration in this example suggests either 0, +1, or −1 transformations, any number of transformations is possible.

Figures 4.13, 4.14 and 4.15 plot the control responses for the ID3, PLS1 and BP16 transformations, respectively, on the Flag task. The results for ID3 and PLS1 follow a similar pattern. Until enough points are available for estimating a parabola, the Select-Transformation procedure recommends a move of one positive transformation. This phase corresponds to the initial rise in the control response. Eventually, adding additional knowledge degrades performance and allows estimation of a parabola for the peak region of the performance response. This phase corresponds to the first decline in the control response. Then, the parabola recommends a negative transformation back to the identified peak, which corresponds to the next rise in the control response. After arriving at the peak, the parabola recommends zero transformations, triggering termination of the control loop. The PLS1 plots in Figure 4.14 missing the trough in the control response indicate that the performance response has little or no degradation in performance.

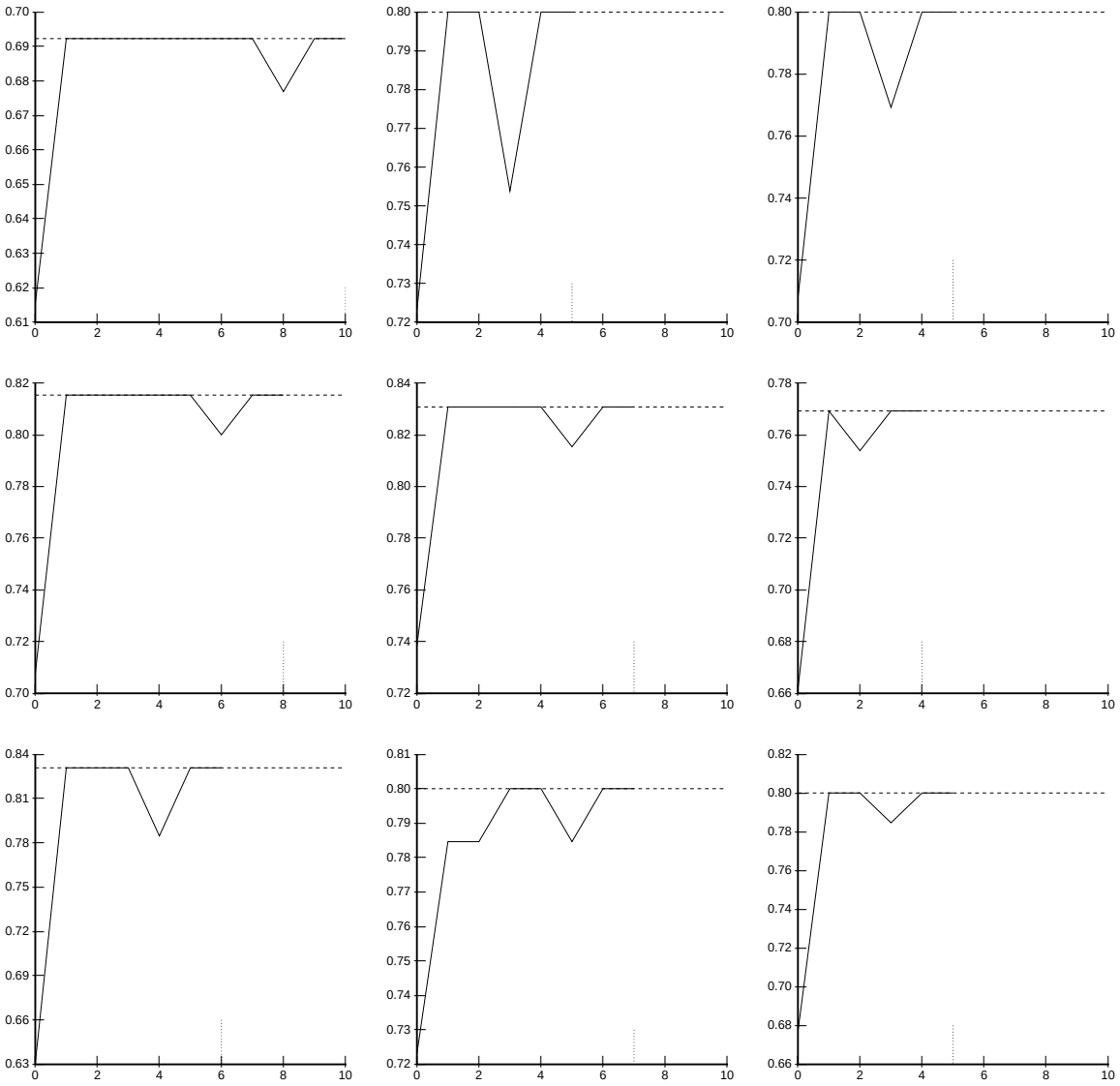

Figure 4.13: MBAC adaptation of ID3 on Flag. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (ID3 splits/unsplits). The vertical dotted line marks the terminating control iteration.

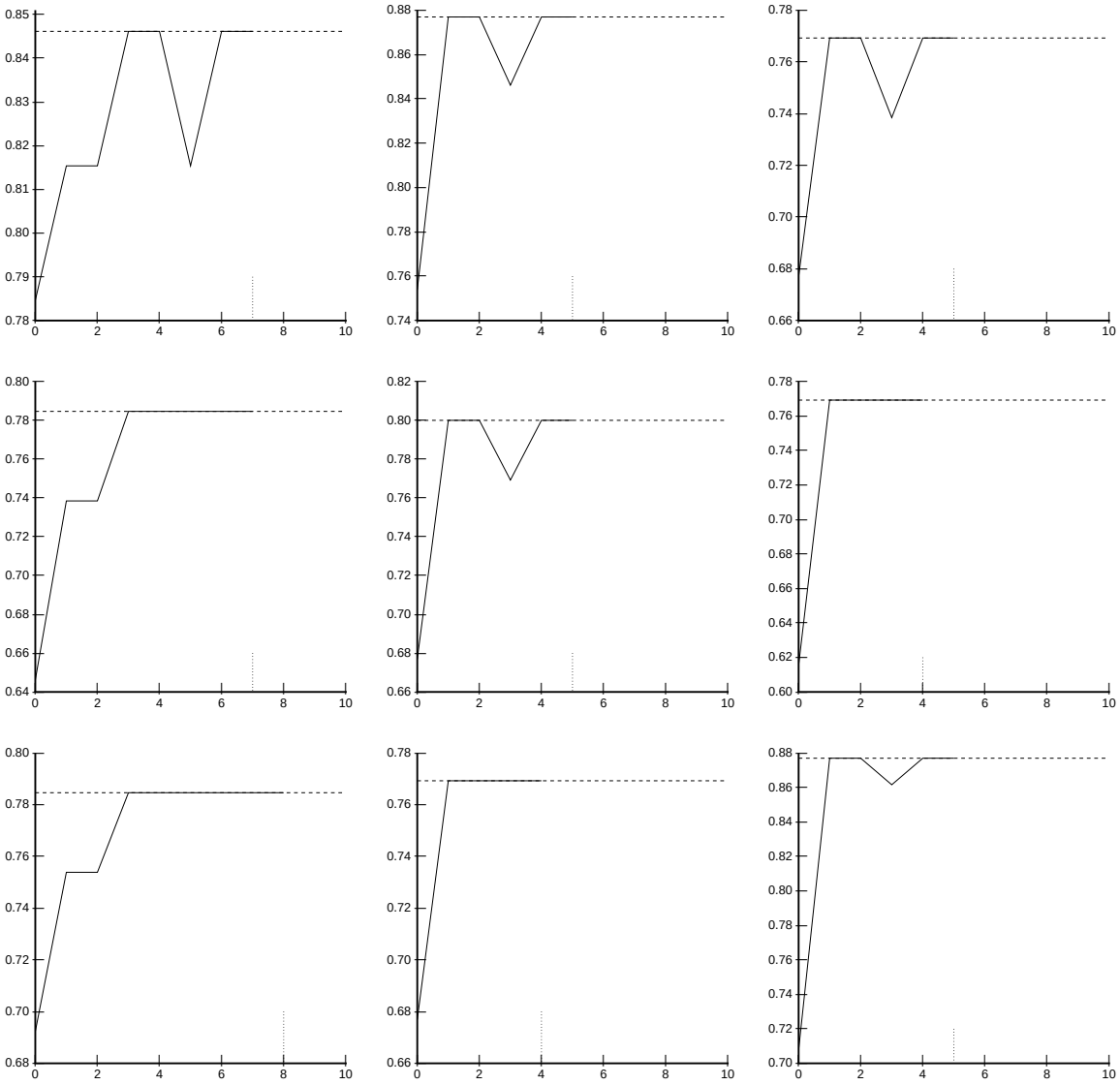

Figure 4.14: MBAC adaptation of PLS1 on Flag. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (PLS1 splits/unsplits). The vertical dotted line marks the terminating control iteration.

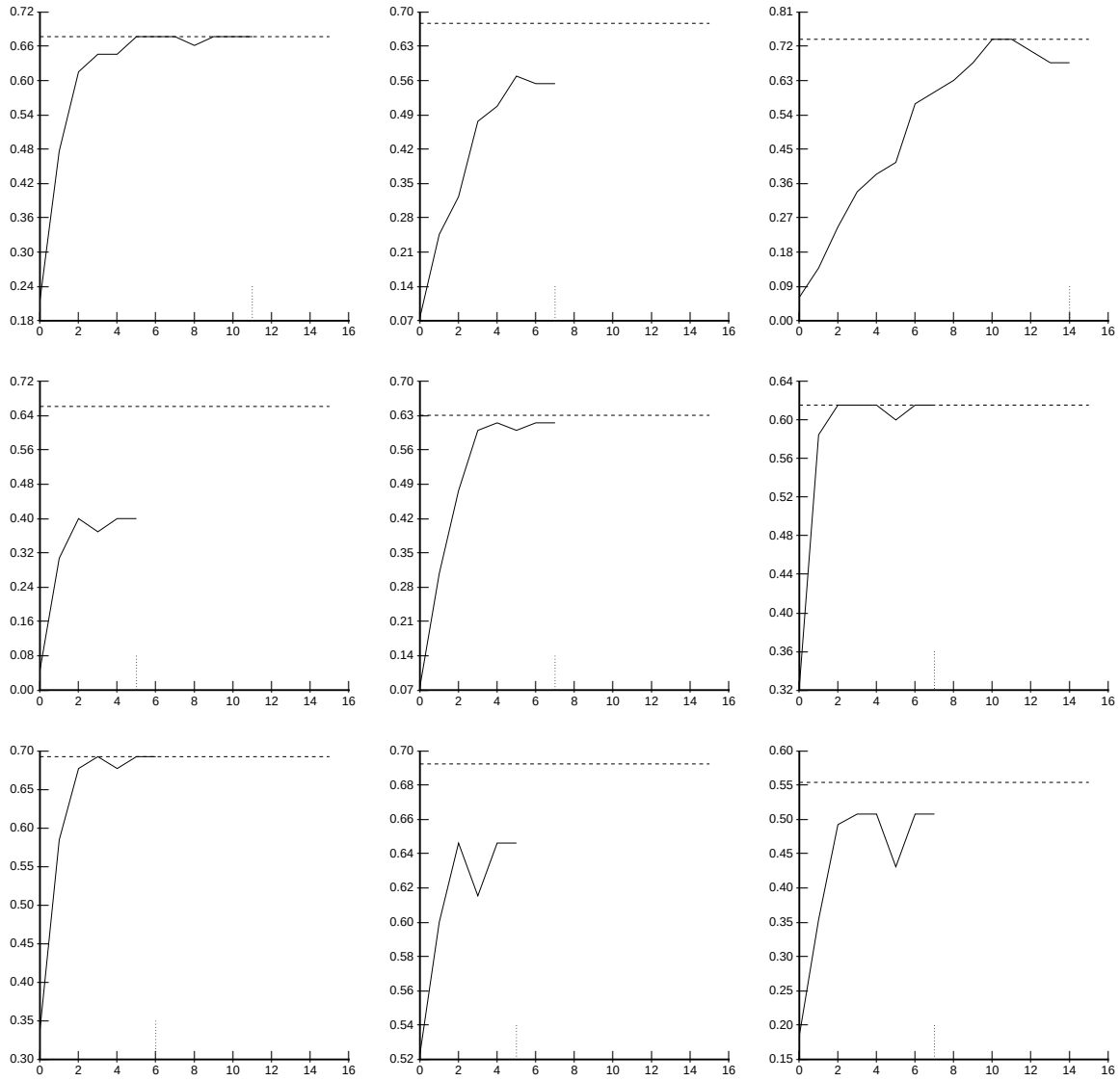

Figure 4.15: MBAC adaptation of BP16 on Flag. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (multiples of five BackProp cycles/uncycles). The vertical dotted line marks the terminating control iteration.

The plots for BP16 in Figure 4.15 reveal less successful behavior of the adaptive control procedure. Three of the plots follow the trend of the ID3 and PLS1 plots (upper-left, center-right and lower-left). Although the remaining control responses also reflect this trend, they stablize at a point beneath the peak of the performance response. The reason for this behavior is the existence of a local peak at the beginning of the performance response before the optimal peak. The parabola estimator fits the local peak, and the control response stabilizes at this peak. This behavior is mainly due to the locally erratic performance responses obtained from the back-propagation method.

If all three transformations were given to the MBAC procedure, the control responses would be the same. MBAC would select a transformation at random and execute the control loop until the model recommends zero transformations. Then, the next transformation takes control. When all transformations recommend zero transformations, MBAC selects the transformation with the least error and most certainty. MBAC uses the knowledge corresponding to this transformation.

The results of Experiment 4 indicate that in most cases MBAC's adaptive control algorithm achieves an accurate model of the performance response during the initial dynamics of the MBAC approach. The model allows MBAC to control the amount of learned knowledge to reside at the peak of the performance response and avoid the generation of low utility knowledge.

The main deficiency in the initial dynamics of the MBAC approach is the sensitivity of the parabola estimation procedure to variations in the data sampled from the performance response. Several solutions exist for this problem. As discussed in Section 3.5.4, one solution is to change the stopping criterion of the adaptive control loop by removing the test for zero transformations. This change forces MBAC to continue acquisition of samples from the performance response and improves the chance that MBAC will see beyond the local peak.

A second solution involves a less precise method for identifying the initial degradation by maintaining a window of performance response sample points for computing the gradient of the response. Only after the gradient begins to decline does the parabola estimator fit a parabola to the sampled points. The addition of windowing to the MBAC parabola estimation procedure has only a small effect. Figure 4.16 shows the upper-middle control response from Figure 4.15 (BP16 on the Flag task) using a window size of three. In this case, the windowing improves MBAC's performance by allowing the control algorithm to converge on the global peak of the

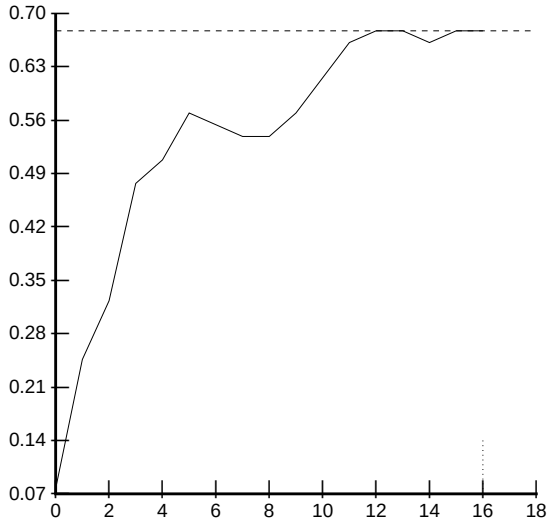

Figure 4.16: The effect of windowing on the upper-middle control response of BP16 on the Flag task. Window size is three.

performance response at the expense of more iterations. However, the window of size three does not change the remaining control responses from Figure 4.15. Attempts to increase the window size also did not help. Larger window sizes eventually cause divergence in the control response, because the window becomes too large to discern the peak.

A third solution to the sensitivity of the parabola estimator is the use of more robust curve-fitting techniques. Press *et al.* [1986] describe statistical techniques for improving the robustness of the curve fitter. However, these techniques increase the cost of computing the models. The fourth solution to the problem utilizes other models from known tasks to suggest a possible peak location in the new task model. The next section investigates this solution.

4.6 Experiment 5: Task Transfer

The previous experiment illustrates how MBAC adapts the models from no prior knowledge of the task domains. However, after acquiring models for several task domains, MBAC can transfer the recommendations of these known models to the unknown models of new tasks. Experiment 5 illustrates how task transfer improves the initial adaptation of models for new tasks and, in some instances, improves the model's final performance. The next section de-

scribes the implementation of MBAC's task transfer procedure, and Section 4.6.2 presents the experimental method and the results of applying task transfer to the scenario of Experiment 4.

4.6.1 Task Transfer

Given a new task, an associated objective, and a set of transformations for achieving the new task objective, MBAC creates a new model instance for each transformation. The new model instances for the new task express the relationship between the performance dimension of the objective and the transformations. Experiment 4 shows how MBAC can acquire initial data for this new model, but also illustrates some shortcomings of the approach. One solution to these problems is to perform some initial experimentation in the new task domain; that is, prime the new models with samples from the unknown performance responses of the new task. The difficulty with this approach is determining the amount of experimentation.

Task transfer addresses this difficulty by using the models of known tasks to recommend the number of transformations to make in the new task. For example, suppose MBAC has models for the ID3, PLS1 and BP16 transformations in the Breast-Cancer, Flare and Voting task domains. Now, MBAC receives the new Flag task along with the objective for accuracy and creates three new models, one for each transformation. When using task transfer to help build a new model for, say, ID3 in the new task domain, MBAC first determines the recommendation for the number of ID3 transformations needed to achieve the new objective in the three known task domains. MBAC uses the average of these recommendations as an initial recommendation for the new model. Then, MBAC performs the recommended number of transformations one by one, recording performance after each transformation and storing these samples of the performance response in the set of data for the new model. The task transfer procedure repeats this process for each new model. The result is a set of new models with enough performance response samples to make better initial control decisions than a model with fewer samples.

The task transfer procedure differs only slightly from making a random number of initial transformations to prime the models. However, this difference is significant, because without a reasonable estimate of how many transformations are needed to accurately represent the performance response curve, there is no guarantee that a random number of transformations would suffice. The average recommendation from similar models provides a better estimate

```
procedure Task-Transfer (task, objective, models)
begin
  foreach model in models do
     decision = Task-Transfer-Decision(task, objective, model)
     for i = 1 to decision-move(decision) do
        model-move(model) = 1
        Perform-Transformation(model) ;see Figure 4.11
end

procedure Task-Transfer-Decision (task, objective, model)
begin
  dimension = objective-dimension(objective)
  threshold = objective-threshold(objective)
  decisions = {}
  foreach m in *mbac-models* do
     if model-task(m) ≠ task and
       model-dimension(m) = dimension and
       model-transformation(m) = model-transformation(model)
     then decisions = decisions + Parabolic-Decision(m, threshold) ;see Figure 4.7
  return(mean(decisions))
end
```

Figure 4.17: Task transfer procedure.

than random recommendations. Therefore, the task transfer procedure utilizes the models for known tasks to benefit the initial adaptation of models for new tasks.

When using task transfer, the MBAC procedure in Figure 4.9 calls the Task-Transfer procedure after building the initial models, but before entering the control loop. Figure 4.17 outlines the Task-Transfer procedure. For each of the transformations' models, Task-Transfer calls Task-Transfer-Decision (described below) for the average number of transformations recommended by the same transformation for other tasks. The procedure then performs the recommended number of transformations one at a time by calling the Perform-Transformation procedure in Figure 4.11. The model data now has a greater number of points sampling the performance response, and the model index already resides at the number of transformations estimated to achieve the desired objective.

The Task-Transfer-Decision procedure (also in Figure 4.17) returns the average decision for models from other tasks with the same transformation as the given model. For each of the models in the set of models known to MBAC (**mbac-models**), the procedure determines if

the model corresponds to a task other than the given task and has the same dimension and transformation as the given model. If the model satisfies these constraints, Task-Transfer-Decision calls the Parabolic-Decision procedure in Figure 4.7 to obtain the model's control decision for achieving the objective. The Task-Transfer-Decision procedure retains each decision and returns the average.

4.6.2 Method and Results

Experiment 5 follows a similar method to Experiment 4. First, the method initializes the set of known models (**mbac-models**) to the models used in experiments $1-3$ for the Breast-Cancer, Flag and Flare task domains and the ID3, PLS1 and BP16 transformations. Then, the method calls the MBAC procedure (augmented with the call to Task-Transfer) with the Flag task, the performance objective of classification accuracy = 1.0, and one of the three transformations. The method retains the performance of the initial hypothesis, the performance after the initial task-transfer control decision, and the performance after each iteration of the adaptive control loop. The experimental method repeats the MBAC call nine times with the same nine randomly selected training and testing sets used in Experiment 4.

Figures 4.18, 4.19 and 4.20 plot the control responses for the ID3, PLS1 and BP16 transformations, respectively, on the Flag task. See Figure 4.12 for an explanation of the control response. Comparison of these control responses to those of Experiment 4 reveal the desired effect of task transfer. The ID3 control responses in Figure 4.18 show that the model, primed with task transfer, achieves the best possible performance in one iteration of the MBAC adaptive control loop.

Similarly, the PLS1 control responses in Figure 4.19 show that task transfer reduces the number of control-loop iterations. However, the upper-left control response in Figure 4.19 indicates that the performance response samples provided by task transfer are still insufficient to accurately model the peak, because more iterations are necessary to find the point of degrading performance that identifies the peak. In this case, the recommendation from task transfer underestimates the number of samples necessary to identify the performance response. The same argument holds for the other control responses in Figure 4.19 that require more than two control iterations to reach maximum performance.

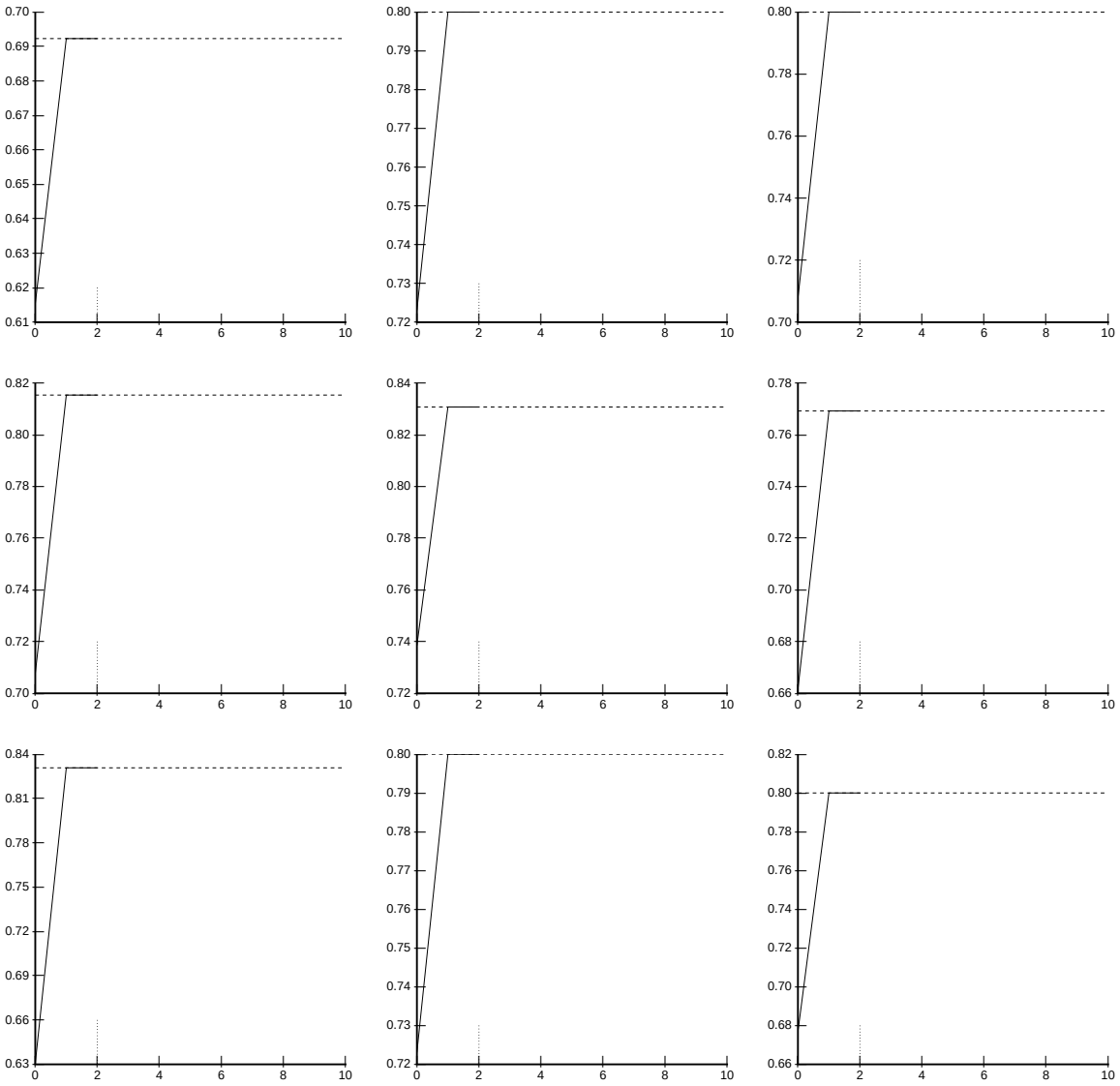

Figure 4.18: MBAC adaptation of ID3 on Flag with task transfer. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (ID3 splits/unsplits). The vertical dotted line marks the terminating control iteration.

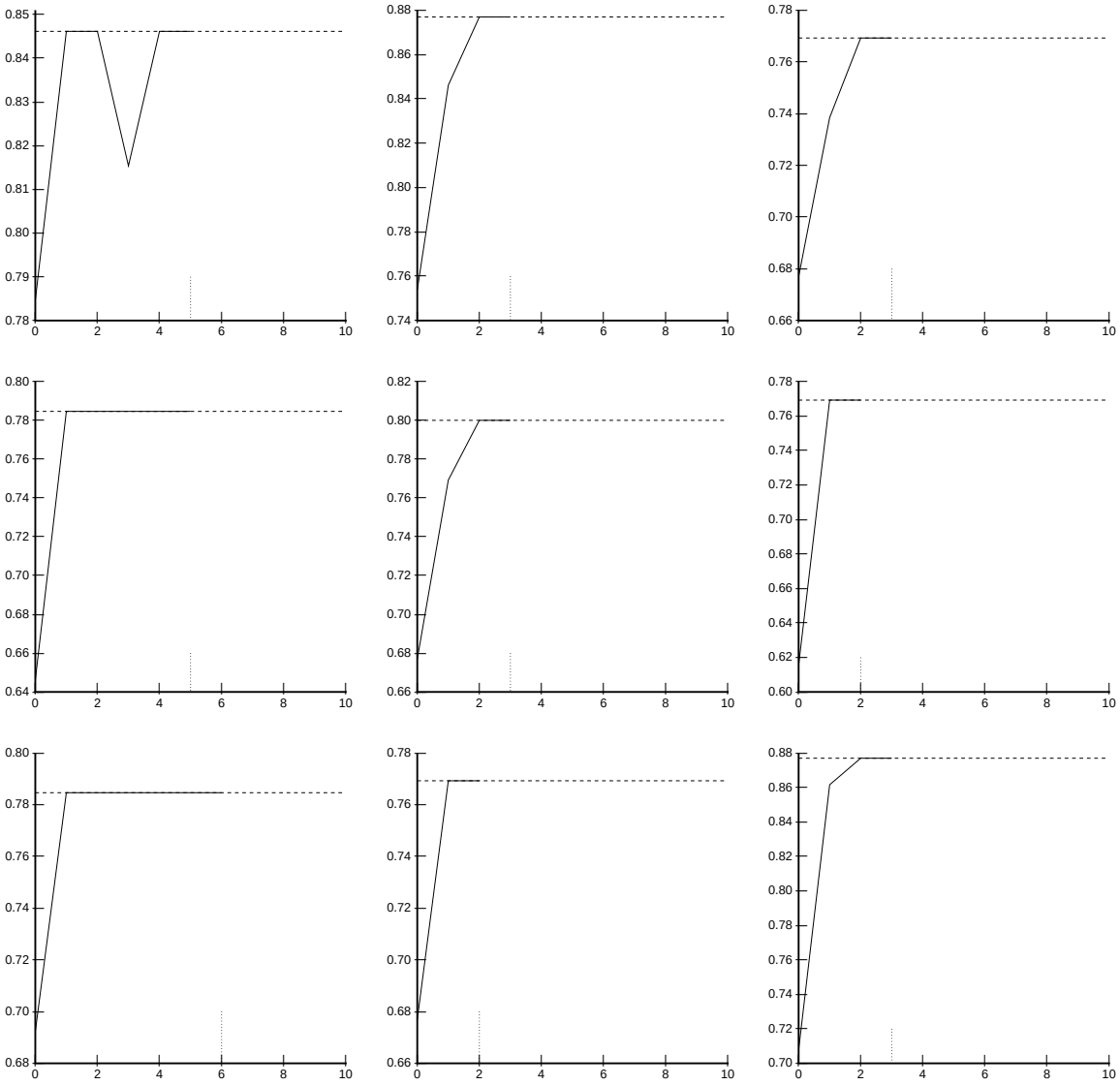

Figure 4.19: MBAC adaptation of PLS1 on Flag with task transfer. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (PLS1 splits/unsplits). The vertical dotted line marks the terminating control iteration.

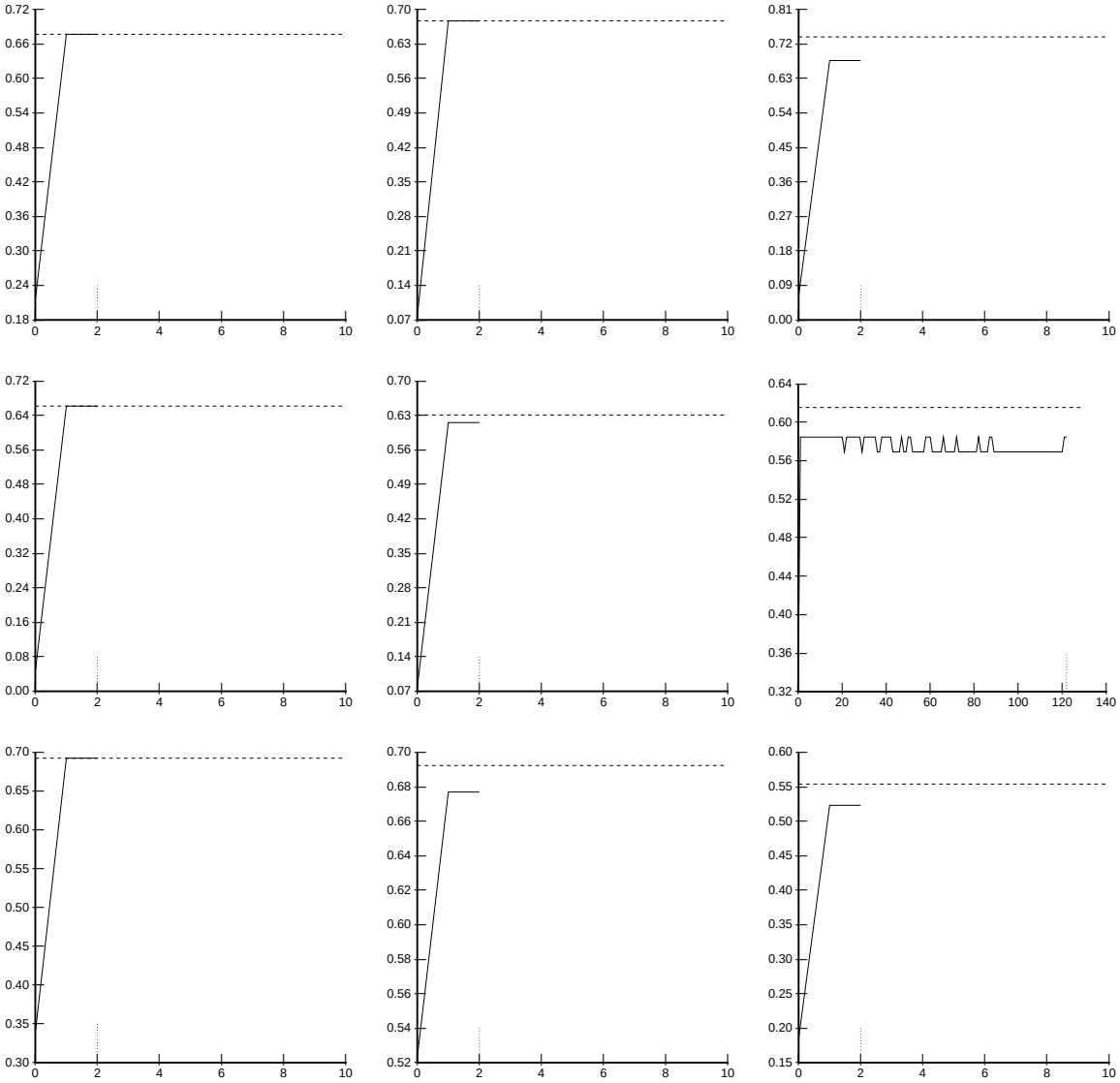

Figure 4.20: MBAC adaptation of BP16 on Flag using task transfer. The y-axis of the plots measures the classification accuracy, and the dashed line represents the maximum achievable accuracy. The x-axis indicates the number of control iterations of the MBAC adaptive control loop. Each iteration represents one or more transformations (multiples of five BackProp cycles/uncycles). The vertical dotted line marks the terminating control iteration.

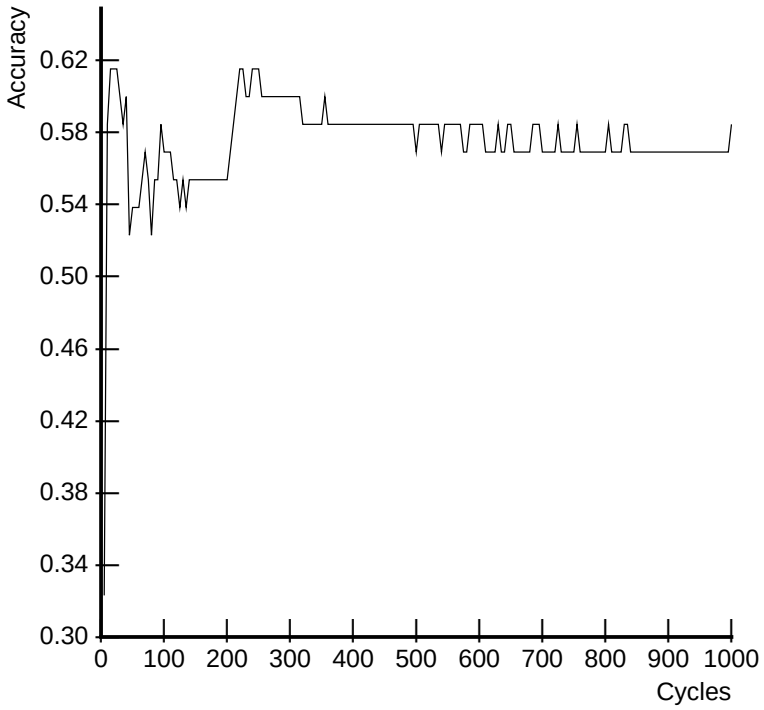

Figure 4.21: Anomalous behavior of one individual performance response for the BP16 transformation on the Flag task. The two peaks at 220 and 245 cycles have the same performance value as the initial peak at 20 cycles.

The BP16 control responses in Figure 4.20 show the most improvement provided by task transfer, but also reveal a possible disadvantage. Compared to the BP16 control responses without task transfer in Figure 4.15 of Experiment 4, the inclusion of task transfer reduces the number of control-loop iterations necessary to achieve the performance levels in Figure 4.15. Furthermore, the final performance levels in three of the task-transfer control responses exceed the corresponding final performance levels obtained in Experiment 4. This improvement is due to the additional performance response samples added by task transfer, which allow the parabola estimator to ignore local peaks and identify the global behavior of the performance response. This result also suggests that MBAC should take more samples when task transfer is unavailable.

The right-center control response in Figure 4.20 reveals a possible disadvantage of using task transfer. In this case, the number of transformations recommended by the task transfer procedure overestimates the number necessary to perceive the global behavior of the performance

Table 4.6: Comparison of MBAC approach without task transfer (mbac) and with task transfer (transfer) to the best possible performance (peak). The entries measure classification accuracy.

	Breast Cancer			Flag		
	peak	mbac	transfer	peak	mbac	transfer
ID3	0.699	0.641	0.641	0.793	0.793	0.793
PLS1	0.741	0.700	0.700	0.808	0.808	0.808
BP16	0.486	0.340	0.458	0.643	0.565	0.628
	Flare			Voting		
	peak	mbac	transfer	peak	mbac	transfer
ID3	0.823	0.812	0.812	0.971	0.967	0.967
PLS1	0.821	0.815	0.815	0.967	0.964	0.964
BP16	0.520	0.399	0.469	0.918	0.901	0.913

response. Normally, the addition of samples from further down the performance response has no effect on the parabola estimation procedure, because the procedure discards the samples beyond the observed peak. However, the situation is complicated by the anomalous performance response shown in Figure 4.21. The anomaly resides in the occurrence of three peaks having identical performance at 20, 220 and 245 cycles. The same control response in Experiment 4 (Figure 4.15) converges to the first peak, because MBAC without task transfer never observes the later peaks. However, since the task transfer recommendation for BP16 is 390 cycles, the resulting model includes the two later peaks. Using these samples, the parabola estimator is unable to find a concave-down parabola to fit the data. Therefore, the Parabolic-Decision procedure returns *nil*, and the Select-Transformation procedure continues to recommend a single forward transformation. The control response follows the performance response in Figure 4.21 beyond the 390 initial cycles and terminates after reaching the end of the performance response. Although this behavior is the result of an anomalous performance response, the occurrence of this situation indicates that task transfer may have detrimental effects and that the parabola estimation procedure may benefit from a more intelligent sample filter placing greater weight on the initial peak in the presence of multiple equivalent peaks.

Experiment 5 shows how MBAC with task transfer reduces the number of control iterations and, in some cases, improves the final performance achieved by the transformed knowledge. Table 4.6 compares the performance of the MBAC approach with and without task transfer to the best possible (peak) performance. Each entry represents the average over ten performance

Table 4.7: Cost comparison between MBAC without task transfer (mbac) and MBAC with task transfer (transfer). The entries measure the number of forward transformations.

	Breast Cancer		Flag		Flare		Voting	
	mbac	transfer	mbac	transfer	mbac	transfer	mbac	transfer
ID3	30	33	7	7	68	68	5	5
PLS1	12	12	3	3	11	11	5	5
BP16	13	80	5	78	74	130	7	57

responses. Entries in a task's *transfer* column result from using the other three tasks for performing task transfer. For the ID3 and PLS1 transformations, the performance of MBAC without task transfer and MBAC with task transfer is identical. The identical performance indicates that MBAC without task transfer is still a useful method for these transformations. MBAC without task transfer has less success for the BP16 transformation, because the BP16 performance responses have a greater number of local peaks that can confuse the parabola estimator. Adding task transfer improves the performance by forcing the control procedure to collect more samples from the performance response. Even with task transfer, MBAC converged to a performance level beneath the peak in all but two cases (ID3 and PLS1 on the Flag task). This discrepancy reiterates the need for more robust curve-fitting techniques (see Section 4.5.2).

Although task transfer improves the performance of MBAC, the improvement incurs a cost. One measure of MBAC's cost is the number of transformations made during the execution of the adaptive control algorithm. Because the cost of a forward transformation for ID3, PLS1 and BP is much higher than the cost of a reverse transformation, the cost measure is the number of forward transformations. Although task transfer reduces the number of control iterations, this measure may not reflect the true cost, because each iteration can involve multiple transformations. Table 4.7 shows the number of forward transformations made by the MBAC adaptive control algorithm with and without task transfer. For ID3 and PLS1, task transfer incurs no extra cost (except for ID3 on the Breast Cancer task), but provides no performance improvement. The performance improvement provided by task transfer for BP16 does incur a cost; however, a similar increase in cost would be necessary for any more robust technique due to the need for further sampling to avoid fitting local peaks.

4.7 Summary

The two components of the general utility problem addressed by this thesis are the generation of low-utility knowledge and the application of inappropriate learning methods. The MBAC approach avoids low-utility knowledge by controlling individual learning methods, and avoids inappropriate learning method application by selecting the methods most likely to achieve desired performance in the task domain. The experimental results of this chapter indicate that MBAC is a valid approach to the general utility problem.

Experiment 1 shows that the parabolic model is superior to the rote or nearest-neighbor model. The parabolic model fits the peak of the performance response more closely than the rote or nearest-neighbor over several different knowledge transformations and tasks. Furthermore, the distance between the actual peak performance and the performance attained using the parabolic model is within the standard deviation of the model. Based on this evidence, MBAC adopts the parabolic model.

MBAC also requires the ability to estimate the certainty of the model. Experiment 2 compares three different certainty estimators: standard deviation, normalized standard deviation, and model probability. Results show that the standard deviation certainty estimate out-performs the other two in terms of correlation to model accuracy and best transformation ordering. Therefore, MBAC adopts the standard deviation as the estimate of model certainty.

Equipped with a model for the performance response and a certainty estimate for the model, MBAC must now select an appropriate transformation and amount of learned knowledge according to the performance objectives and task domain. Experiment 3 shows that the transformation selection mechanism described in Section 4.4.1 performs well at choosing the best transformation.

The first three experiments show MBAC's behavior after considerable sampling of the performance response curve. Experiment 4 illustrates the initial dynamics of MBAC starting with no samples from the performance response. Results show that MBAC quickly acquires a model of the performance response, but the model may be overly sensitive to local peaks in the performance response. Experiment 5 investigates a solution to this problem that improves the initial dynamics of Experiment 4 by transferring recommendations from known tasks to improve the initial decisions made in new task domains. Task transfer reduces the number of

control iterations and, in some cases, improves the final performance of the MBAC adaptive control procedure.

The experiments of this chapter all use a performance threshold higher than that achievable by the available transformations. The use of this threshold does not detract from the robustness of the MBAC approach, because most performance objectives for learning methods concentrate near the peak of the performance response. Therefore, the experiments confirm the ability of MBAC to adaptively and accurately model the peak of the response and use this model to control the application of multiple transformations and the generation of low utility knowledge.

Chapter 5

Related Work

Many of the early learning systems attempted to adapt a model of the performance element in order to control the learning. Buchanan *et al.* [1978] provide a survey of such systems and model the systems as an instance of the adaptive control loop in Figure 1.2. The emphasis of the survey distinguishes the systems based on their expression of the relationship between performance and learned knowledge. The systems use this expression to critique the knowledge learned by the system. Dietterich and Buchanan [1983] provide an analysis of the controlling element (the critic) in many of these systems. As with other methods analyzed in this work, these early systems depend on knowledge of the performance environment; whereas, the MBAC approach attempts to reduce this dependence.

After much work on these domain knowledge-sparse approaches, machine learning research then moved towards more domain knowledge-intensive methods [Dietterich *et al.*, 1982]. However, as Chapter 2 demonstrates, both knowledge-sparse and knowledge-intensive systems need the ability to control themselves in the presence of the general utility problem. The related research discussed in this chapter falls into three categories: control of the utility of learned knowledge, control of multiple learning methods, and adaptive control theory. Since the related work deals with controlling some element of the learning process, this chapter casts the systems in terms of their control method and compares the method with the MBAC approach.

5.1 Utility Control

Chapter 2 describes two methods for controlling the utility of learned knowledge in analytical learning. Section 2.3.1 describes the Prodigy system [Minton, 1988a], which retains learned knowledge having high utility with respect to the task. Utility is the difference between the savings provided by the knowledge and the cost of retaining the knowledge. If empirical estimation of utility yields a negative value for some piece of knowledge, Prodigy discards this knowledge. Prodigy differs from MBAC in that Prodigy determines *which* knowledge to learn; whereas, MBAC determines *how much* knowledge to learn. Prodigy's ability to determine which knowledge to learn derives from a theory of the effects that knowledge has on the performance element. MBAC lacks such an analytical theory, relying instead on an empirical model.

Section 2.3.2 describes the Soar system, which limits the expressiveness of learned knowledge to avoid generation of low utility knowledge [Tambe and Rosenbloom, 1989]. Soar limits expressiveness by constraining each chunk of knowledge to have linear match cost. This approach does not control the utility, because the higher number of less-expressive chunks will also eventually degrade performance.

The following sections describe other systems that control the utility of learned knowledge. The approaches span both analytical and empirical learning paradigms.

5.1.1 MetaLEX

The MetaLEX system [Keller, 1987a; Keller, 1987b] adapts a set of problem solver control knowledge in order to achieve given performance objectives. The search control knowledge represents the concept of a useful move for the problem solver; however, the initial expression of this concept is not operational. Keller defines an *operational* useful-move concept as one that allows the problem solver to satisfy the desired performance objectives. The performance objectives determine the operationality of the concept, just as they determine the utility of knowledge in the MBAC approach. Therefore, MetaLEX is adaptively controlling the utility of the problem solver control knowledge.

Figure 5.1 depicts MetaLEX as an adaptive control loop. The search control knowledge acts as a pruning filter on the set of operators available to the problem solver. Guided by the search control knowledge, the problem solver performs a breadth-first search for the solution to each

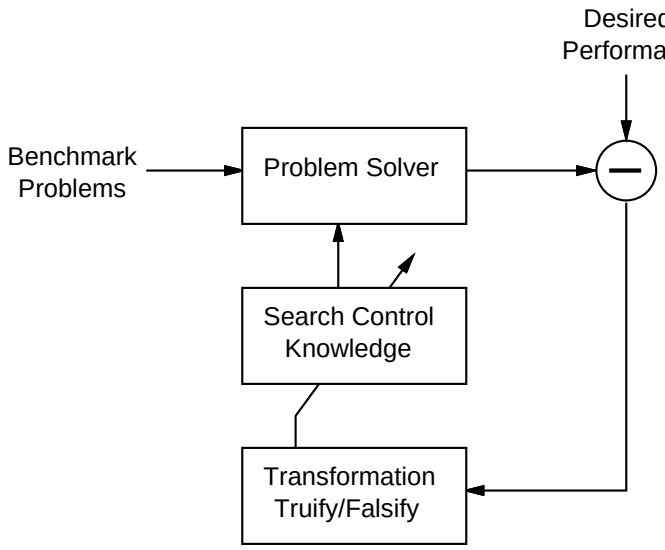

Figure 5.1: MetaLEX as Adaptive Control

benchmark problem. MetaLEX measures the percentage of the benchmark problems solved by the problem solver (effectiveness) and the cumulative CPU time of solving these problems (efficiency). If the effectiveness and efficiency performance measures do not satisfy the desired performance objectives, MetaLEX uses contextual knowledge from the problem solutions to select a transformation for modifying the search control knowledge. This process continues until the search control knowledge allows the problem solver to satisfy the desired performance objectives on the set of benchmark problems.

MetaLEX's task domain is symbolic integration. Each benchmark problem is an integral (e.g., $\int \sin(x)\,dx$). Each operator transforms an integral into an expression without an integral, or a simpler integral (e.g., $\int \sin(x)\,dx \rightarrow -\cos(x)$). MetaLEX contains thirty such operators. The goal state for each problem is an expression containing no integral signs.

5.1.1.1 Search Control Knowledge

The knowledge in MetaLEX is the description of the useful-move concept. At each node in the search space for a solution to a benchmark problem, the problem solver uses the useful-move concept to prune the set of possible operators applicable at this node. Figure 5.2 shows the non-operational description of the useful-move (USEFUL) concept. A move is useful if execution of

```
USEFUL(move) ⇔
   (LET ((succ (EXECUTE move)))
     (OR (SOLVED succ)
         (SOLVABLE succ (- *maxdepth* (MOVEDEPTH move)))))

where

SOLVED(state) ⇔ (NOTMATCH state 'ʃ)

and

SOLVABLE(state depth) ⇔
   (AND (> depth 0)
        (OR (FOR binding IN (BINDINGS 'OP1 state)
              (LET ((succ (APPLY 'OP1 binding)))
                (OR (SOLVED succ)
                    (SOLVABLE succ (- depth 1)))))
            (FOR binding IN (BINDINGS 'OP2 state)
              (LET ((succ (APPLY 'OP2 binding)))
                (OR (SOLVED succ)
                    (SOLVABLE succ (- depth 1)))))
            ...))
```

Figure 5.2: Non-operational search control knowledge for the useful-move concept.

the move reaches the goal state (SOLVED) or if the move is along the path to the goal state (SOLVABLE).

5.1.1.2 Transformations

MetaLEX transforms the knowledge of Figure 5.2 using the TRUIFY and FALSIFY transformations. These transformations replace predicate-valued subexpressions of the useful-move concept with the constant TRUE or FALSE. MetaLEX also uses the reverse of these transformations: UNTRUIFY and UNFALSIFY. In practice, MetaLEX transforms only the disjuncts of SOLVABLE in one of three ways: falsifying the entire disjunct (operator rarely useful), falsifying the internal SOLVED subexpression (operator rarely reachs a goal state), or truifying the internal LET subexpression (operator almost always useful). The TRUIFY transformation generalizes the useful-move concept by recommending more operators; whereas, the FALSIFY transformation specializes the useful-move concept by recommending fewer operators.

5.1.1.3 Performance Objectives

The performance objectives in MetaLEX constrain the performance measures of efficiency and effectiveness. MetaLEX measures efficiency as the time (in CPU seconds) needed by the problem solver in attempting to solve the set of benchmark problems. Effectiveness is the percentage of the benchmark problems actually solved by the problem solver. MetaLEX defines the performance objectives as

$$\text{Efficiency} < t$$
$$\text{Effectiveness} \leq p.$$

where t and p are user-supplied thresholds on the performance measures.

5.1.1.4 Model

MetaLEX models the relationship between performance and knowledge transformations with qualitative heuristic trends. Figure 5.3 depicts these trends as performance response curves. Point a on the horizontal axis of the curves represents the point at which the useful-move concept is completely truified or falsified. Point c on the efficiency axis represents the efficiency (CPU time) of the problem solver using the initial non-operational useful-move concept. Note that the curves are only approximations of the heuristic trends identified in MetaLEX [Keller, 1987b].

For the efficiency versus TRUIFY response curve, truifying subexpressions of the useful-move concept generally improves efficiency (lowers CPU time) until the model begins to recommend non-useful moves. At this point the curve moves upward until the entire useful-move concept is truified at point a. Point a represents the problem solver's efficiency without the use of search control knowledge. Note that the inverse of this response curve resembles the general utility problem trend of Figure 2.1. For the efficiency versus FALSIFY response curve, falsifying subexpressions of the useful-move concept generally improves efficiency. At point a, the useful-move concept recommends no operators, and the problem solver does nothing.

For the effectiveness versus TRUIFY response curve, truifying subexpressions of the useful-move concept has no effect on effectiveness, because TRUIFY only increases the number of operators considered useful. For the effectiveness versus FALSIFY response curve, falsifying subexpressions of the useful-move concept generally degrades effectiveness, because FALSIFY

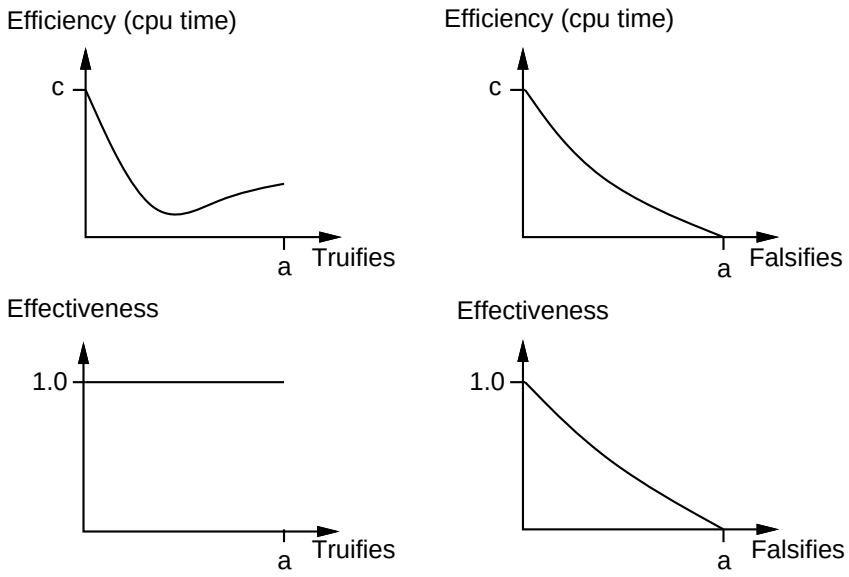

Figure 5.3: Performance responses for MetaLEX.

only decreases the number of operators considered useful. At point a, the useful-move concept recommends no operators, and the problem solver solves none of the benchmark problems.

MetaLEX adapts these trends according to statistics collected during the problem solver's attempt to solve the benchmark problems. When performance falls below the desired objectives, MetaLEX uses these adapted trends along with contextual knowledge from problem-solving traces to select the best transformation (TRUIFY or FALSIFY) and subexpression of the useful-move concept that will move performance closer to the objectives. When performance estimates indicate achievement of the objectives, the hill-climbing transformation procedure ceases and the problem solver attempts to solve the benchmark problems using the transformed useful-move concept. If performance does not meet the objectives, then MetaLEX uses the same information to assess blame on recent transformations, which are then undone until estimated performance is again within the objectives. This procedure continues until the problem solver can solve the benchmark problems within the desired performance objectives.

5.1.1.5 Comparison to MBAC

Despite the similarity to the adaptive control structure of MBAC, MetaLEX takes the opposite end of the generality/contextual-knowledge spectrum. At MBAC's end of the spectrum, the approach is general across several learning transformation methods, knowledge representations,

performance dimensions, tasks and performance elements. Therefore, MBAC can expect little contextual-knowledge help in transforming the knowledge to achieve the performance objectives. At the other end of the spectrum, the MetaLEX method applies to only two specific transformations, one knowledge representation, two performance dimensions, and one performance element. Although MetaLEX uses only one task to demonstrate the method, there is nothing to prevent other tasks.

Due to the specific performance environment, MetaLEX derives much knowledge from the context of the environment to aid in the selection of a proper transformation. The models in Figure 5.3 assume execution of only one transformation, TRUIFY or FALSIFY. Because MetaLEX intermixes applications of the two transformations, the contextual knowledge integrates the TRUIFY and FALSIFY models for each performance dimension. The MBAC approach is unable to intermix transformations in this way. The contextual information from solution traces and statistics generated during solution of the benchmark problems helps MetaLEX select *what* search control knowledge to transform, as well as *how* to transform this knowledge. The performance objectives determine *how much* to transform the search control knowledge.

Keller emphasizes the need for explicit performance objectives in MetaLEX in order to more precisely define the operationality of knowledge. MBAC advocates a more explicit performance *environment*, including the performance element, transformations and knowledge, as well as the performance objectives. The element of MBAC that permits this further explicitness is the model of the general utility problem trend of Figure 2.1. The independence of this model from contextual knowledge allows MBAC to work in a more explicit, and therefore more general, performance environment.

5.1.2 Composer

The Composer system [Gratch and DeJong, 1991] controls the utility of learned knowledge (set of control rules) by adding a control rule to the existing set of control rules only if the new rule has high conditional utility with respect to the current set of control rules. Composer measures the conditional utility of a control rule r with respect to a set of control rules R as

$$\text{Utility}(\{r\} \cup R \mid \emptyset) = \text{Utility}(R \mid \emptyset) + \text{Utility}(\{r\} \mid R)$$

Composer uses the Prodigy system to generate candidate control rules and maintains empirical estimates of each rule's condition utility with respect to the current control strategy and a confidence bound on this estimate. When a control rule has a significantly positive conditional utility, Composer adds the rule to the control strategy. Appropriate settings for the confidence bounds increases the probability that the additional control rule will add utility to the control strategy. The higher the desired probability, the more examples Composer needs to insure positive utility.

The expression for conditional utility constitutes a formal model relating the knowledge (control rule) to performance. The model depends on knowledge of the performance element and domain theory. Gratch and DeJong's experimentation with Composer indicates that fewer control rules are necessary to improve performance than retained in the Prodigy system. This observation supports the trend of the general utility problem in which peak performance occurs early with smaller amounts of learned knowledge. Like Prodigy, Composer identifies which knowledge positively affects performance at the expense of extracting knowledge about the performance environment.

5.1.3 Minimum Description Length

The minimum description length (MDL) principle [Rissanen, 1989] states that the best theory to infer from a set of data is the one that minimizes the sum of the length of the theory and the length of the data as described by the theory. The power of the MDL approach comes from an appropriate selection for the encoding scheme that converts the theory and data into a string of symbols. An appropriate encoding scheme allows the MDL principle to find the theory that still has high accuracy on unseen data.

Quinlan and Rivest [1989] use the MDL principle to control the utility of induced decision trees. Their encoding scheme for decision trees and data is complex and not described here. Using the encoding scheme and the MDL principle, Quinlan and Rivest are able to identify decision trees of an appropriate size that reduce overfit and maintain utility. The success of this method relies on the encoding scheme to relate the length of the encoded string to the effects of the corresponding decision tree on classification accuracy. Therefore, the encoding scheme approximates a model relating the performance to the complexity of the decision tree (which can be related to the amount of learned knowledge). The benefit of this approach is the

non-empirical nature of the model. Once an appropriate encoding scheme is found, the MDL principle easily selects the best theory.

5.1.4 APU

In addition to the learning paradigms discussed in Chapter 2, the general utility problem also extends to analogical learning. The APU system [Bhansali and Harandi, 1991] synthesizes UNIX shell scripts using derivational analogy [Carbonell, 1986]. Bhansali and Harandi show that the acquisition of base cases for derivational analogy provides a factor of two speedup for the problem solver; however, the results assume a uniform distribution over the examples in the domain. If the distribution is not uniform, then an increasing number of base cases will eventually degrade problem-solving performance.

Bhansali and Harandi propose a measure for evaluating the utility of adding a new base case to the case library. The library indexing scheme associates to each unique feature the cases that contain that feature in their definition. The measure of utility is the change in the average ratio of cases per feature in the library indexing scheme. If the ratio increases, then the added cases are similar to existing cases and have low utility with respect to analogical problem-solving. If the ratio decreases, then the added cases are different from existing cases and may add new knowledge towards the problem-solving domain.

This model of utility uses knowledge of the analogical learning process; namely, similar cases have less benefit than dissimilar cases, because the similar cases add little additional information compared to the cost of storing and retrieving them. The utility measure is similar to that used in the Prodigy system and, therefore, stresses *which* knowledge to learn versus *how much* knowledge to learn.

5.2 Multiple Learning Method Control

The problem of controlling the utility of learned knowledge is difficult enough for a single learning method. However, some systems have attempted the control of multiple learning methods within a single framework. The following sections describe three such systems and compare them to the MBAC approach.

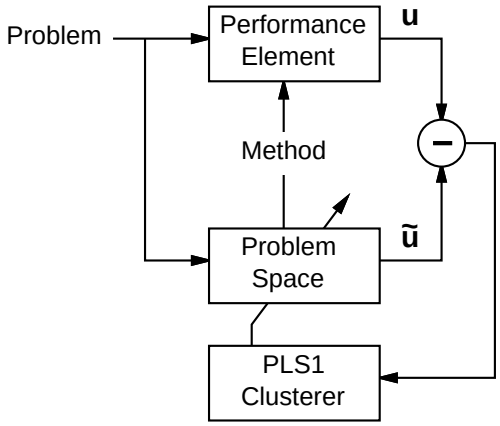

Figure 5.4: VBMS as Adaptive Control

5.2.1 VBMS

The variable bias management system (VBMS) [Rendell *et al.*, 1987b; Rendell *et al.*, 1987a] learns to select an appropriate inductive bias for a learning problem based on the characteristics of the problem. Inductive bias includes choices for hypothesis representation, inductive method, and parameters of the inductive method; however, Rendell *et al.* [1987a] concentrate on VBMS's ability to select an appropriate inductive method. VBMS maintains a *problem space* whose dimensions are the characteristics of the problem. VBMS learns a function over problem space that maps problem characteristics to inductive methods. Therefore, like MBAC, VBMS adaptively controls the use of multiple learning methods.

Figure 5.4 depicts VBMS as an adaptive control loop. Given a learning problem, VBMS selects an appropriate inductive method according to the function over problem space. Associated with each point in problem space is a vector $\tilde{\mathbf{u}}$ containing utility estimates for each inductive method. VBMS selects the method having the highest utility in $\tilde{\mathbf{u}}$. The performance element applies this method to the problem and evaluates the actual utility $\mathbf{u}$ of the inductive method. VBMS then compares the estimated and actual utilities. If the difference is large, VBMS continues to select inductive methods until $\tilde{\mathbf{u}}$ and $\mathbf{u}$ are similar. VBMS then updates the point in problem space with the new $\mathbf{u}$ and invokes PLS1 to refine the problem space region containing the new point. Over time, VBMS divides the problem space into regions of similar utility in order to select inductive methods appropriate to a given problem.

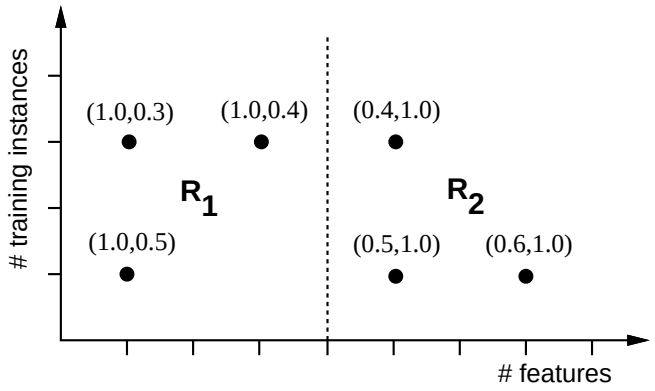

Figure 5.5: Example VBMS problem space showing the utility vectors at each observed point and the two regions R_1 and R_2 resulting from a split made by the PLS1 clusterer.

Experiments by Rendell *et al.* [1987a] involve three inductive learning methods: AQ15 [Michalski *et al.*, 1986], Assistant [Cestnik *et al.*, 1987] and PLS1 [Rendell, 1983] (see also Section 2.2.1.2). The dimensions of the problem space were number of training instances and number of features.

5.2.1.1 Problem Space

The dimensions of the VBMS problem space are the characteristics used to describe the learning problem. Figure 5.5 shows a problem space defined by two characteristics: number of training instances and number of features. The utility vector at each point expresses the normalized effectiveness of the available inductive methods. Figure 5.5 shows utility vectors describing the effectiveness of two inductive methods. Effectiveness represents the performance of the inductive method on the learning problem. Possible dimensions of performance include resource cost of executing the inductive method, classification accuracy of the resulting hypothesis, and comprehensibility of the resulting hypothesis. Values for these measures combine to form a global effectiveness measure. Rendell *et al.* [1987a] used the runtime in CPU seconds as the effectiveness measure. For example, if inductive method I_1 executed in 100 CPU seconds, and inductive method I_2 executed in 200 CPU seconds on the same problem, then the normalized utility vector would be $(1.0, 0.5)$, indicating a preference for I_1.

Based on the utility vectors at each point in problem space, the PLS1 clusterer divides the space into regions of similar utility. For example, Figure 5.5 shows a likely division of problem space for the sample utility vectors. Region R_1 indicates that inductive method I_1 is preferable

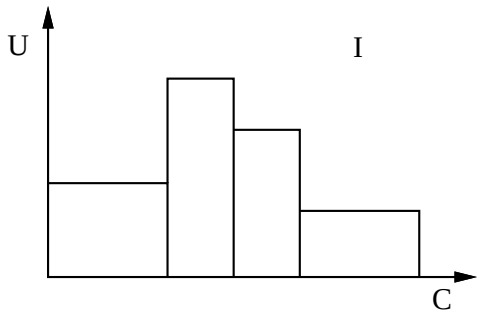

Figure 5.6: VBMS model relating utility U to a particular problem characteristic C for a particular inductive method I.

for problems with few features; whereas, region R_2 indicates that I_2 is better for problems with many features. The division of problem space into regions forms a piecewise constant function mapping problem characteristics to inductive method.

For each region in problem space, VBMS maintains a *region belief table* (RBT), which is the average of the utility vectors of the points within the region. In Figure 5.5 the RBT for R_1 is $(1.0, 0.4)$, and the RBT for R_2 is $(0.5, 1.0)$. When VBMS encounters a new learning problem, the resulting $\tilde{\mathbf{u}}$ in Figure 5.4 is the RBT for the region enclosing the corresponding point in problem space. When the estimated RBT is not sufficiently similar to the emerging utility vector $\mathbf{u}$, VBMS searches for another RBT that more closely matches $\mathbf{u}$.

The function over problem space represents VBMS's model of how transformations (inductive methods) affect performance (utility). For example, Figure 5.6 illustrates a possible model maintained by VBMS for a particular inductive method I, particular problem characteristic C, and utility measure U. The problem space function consolidates these individual models into the multi-dimensional problem space and utility vectors.

5.2.1.2 Comparison to MBAC

Like MBAC, VBMS learns to control the application of learning methods by adapting a model of their performance utility. However, several differences exist between the two systems. One difference is the explicitness of the performance environment. VBMS combines the performance measures for the learning methods into one global utility value. Combining the performance measures prevents a more refined learning method selection based on multiple performance objectives. VBMS also does not explicitly define the stopping criteria for the region belief table

search. Rendell *et al.* [1987b] mention bounds on the rate of utility improvement and resource cost as possible stopping criteria. An explicit representation of the performance environment allows MBAC to better identify the strengths and weaknesses of the available learning methods.

Another difference between the two approaches lies in the expression of the model. The VBMS model is a piecewise constant function relating the utility of a learning method to the characteristics of a learning problem. The MBAC model is a continuous quadratic curve relating performance to the amount of learned knowledge for a single problem. The reason VBMS uses the more general piecewise constant function model is the lack of decomposition of the learning methods. MBAC decomposes the learning methods to a level of granularity sufficient to perceive the performance response trend (see Section 3.4.1). VBMS observes only the final performance of the learning methods and, therefore, requires a more general model to fit the variations in these utility values. Constraining MBAC's model to a quadratic curve avoids the overfitting of the performance response possible with more general models.

One benefit of VBMS over MBAC is the use of problem characteristics as the independent variables to the performance model. The performance of learning methods depends on characteristics of the learning problem. VBMS assumes the user selects proper problem characteristics that convey enough information to detect the relationship to performance for a learning method. MBAC obviates selecting problem characteristics by utilizing the commonality of the performance response curve among learning methods. A promising compromise between the two approaches is to await the derivation of formal models relating performance to problem characteristics (as in the analysis of Section 2.5.1.1). Replacing the empirical model with the formal model yields more accurate estimation of the performance of learning methods.

5.2.2 AIMS

The Adaptive Interactive Modeling System (AIMS) [Tcheng *et al.*, 1989; Tcheng *et al.*, 1991] extends the VBMS approach along several dimensions to form a more robust system for managing inductive bias. AIMS learns to control multiple learning methods by adapting a function that relates performance objectives to inductive bias and using the predicted-optimal bias to select among the competing methods. Figure 5.7 depicts AIMS as an adaptive control loop. Given a learning task, AIMS selects an inductive bias predicted to optimize the performance objectives according to the functional relationship between objectives and bias (i.e., the objective surface

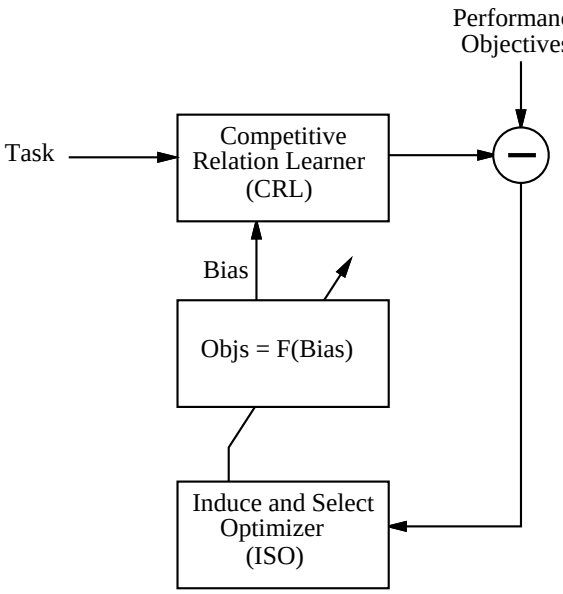

Figure 5.7: AIMS as adaptive control.

over bias space). AIMS uses the selected bias to control the application of learning methods within the competitive relation learner (CRL). CRL outputs a hypothesis whose performance is fed back to the Induce and Select Optimizer (ISO). ISO uses the actual performance value for the currently selected bias to adapt the function relating objectives to bias. AIMS continues adaptation until the hypothesis satisfies desired performance objectives.

5.2.2.1 Competitive Relation Learner

The competitive relation learner (CRL) is a generalized recursive splitting method similar to those described in Section 2.2.1. The difference in CRL is the availability of multiple inductive learning strategies, multiple splitting strategies, and multiple split-evaluation strategies at each node in the tree. Non-generalized splitters commit to one choice for each of these three components. The learning strategies perform operations on the examples at a node in the tree. The available learning strategies are mean, mode, nearest-neighbor, regression (with linear, quadratic, logarithmic and exponential models), and back-propagation neural network. The splitting strategies determine places to divide the region of instance space represented by a node in the tree. Available splitting strategies are to split evenly along each feature dimension, split according to example population, and split along arbitrary hyperplane. The split

evaluation strategies measure and validate the error in CRL's hypothesis. Hypothesis error metrics include average deviation, standard error, entropy and vector difference. Error validation strategies include error on training set, error on testing set, and v-fold cross validation.

The current inductive bias selects a subset of these strategies along with their parameters to build a tree using the generalized recursive splitting technique. CRL tries each strategy at a node in the tree and selects the one yielding the most reduction in the error of the overall hypothesis. CRL continues this process until either the error of the overall hypothesis does not decrease more than a specified threshold, the number of examples at the node is less than a specified threshold, or the time exceeds a specified threshold.

5.2.2.2 Induce and Select Optimizer

In order to reduce the size of the search space considered by CRL, the induce and select optimizer (ISO) uses experience from AIMS to estimate the relationship between performance objectives and inductive bias, and selects a bias that optimizes the performance objectives. Each time AIMS uses CRL (controlled by the current inductive bias) to generate a hypothesis, the performance of the hypothesis provides a new point in the bias space for estimating the objective surface over this space. ISO employs CRL as the inducer. ISO's CRL induces an expression for the objective surface from the examples of performance values at different bias points. ISO's selector uses CRL's suggested objective surface and the examples to select an inductive bias that optimizes the performance objectives. When the user specifies multiple objectives, ISO outputs a set of Pareto optimal (non-dominated) set of biases.

AIMS provides two parameters, novelty and performance, for controlling the selector's use of the examples and the induced objective surface. High novelty urges the selector to ignore the induced objective surface and try biases that are maximally distant from previously-attempted biases. High performance urges the selector to adhere to the induced objective surface and select the point that optimizes that surface. AIMS passes the resulting bias selection to the version of CRL that induces hypotheses for the learning task.

5.2.2.3 Comparison to MBAC

MBAC is similar to AIMS in several ways. First, both systems allow explicitly-defined, multiple performance objectives. AIMS goes further to implement a multi-objective optimizer for dealing

with the tradeoffs among multiple objectives. This ability is not implemented in MBAC. Second, both systems adapt a functional relationship between performance and bias based on experience. Third, both systems use this relationship to control the application of learning methods.

Several differences exists between AIMS and MBAC. First, although both systems adapt a functional relationship between performance and bias, MBAC further constrains the function to be quadratic near the peak of the function as recommended by the performance response trend revealed in Chapter 2. Although AIMS can converge to a quadratic expression of the function, not constraining the function in light of the empirical evidence may allow AIMS to overfit the performance response. However, AIMS has the capability to fit other models to the performance response when the response is not quadratic. Some performance responses not considered in this investigation will have a non-quadratic shape. In this case, AIMS may achieve a better fit to the response than MBAC.

A second difference concerns the dimensions of the bias space. AIMS defines several dimensions to the bias space for controlling the learning, decomposition and evaluation strategies of CRL. MBAC attempts to control only one bias: the amount of learned knowledge. The amount of learned knowledge is more difficult to describe than most parameters used to control learning, but allows the MBAC approach to extend to non-empirical learning paradigms (e.g., analytical learning).

Finally, MBAC maintains separate homogeneous hypotheses for a learning task, while AIMS regenerates a hybrid hypothesis for each inductive bias point. Therefore, MBAC spends less time learning, because a change in recommended bias corresponds to a change in MBAC's existing hypotheses; whereas, AIMS must rebuild the hybrid hypothesis, which may involve re-execution of several learning strategies.

5.2.3 MTL

The Multistrategy Task-adaptive Learning system (MTL) controls the application of multiple learning strategies [Tecuci and Michalski, 1991]. The strategies currently implemented include deduction, analogy, abduction and induction. Along with the learning strategies, MTL maintains facts, generalization hierarchies and rules about the domain. MTL's goal is to derive useful knowledge from the input, where the input and background knowledge are in the form of first-order rules. Given an example, MTL uses the strategies to build a justification tree that

explains the example in terms of the background knowledge. MTL employs the strategies in the order given above. Once the justification tree is generated, MTL generalizes the tree using a technique similar to explanation-based generalization [Mitchell *et al.*, 1986].

Two differences exist between MTL and MBAC. First, MTL relies on a common representation of the knowledge on which each strategy performs some transformation. This dependence on a common representation precludes the use of learners whose hypotheses are not expressed as first-order rules. MBAC maintains a hybrid representation of the knowledge, which allows the learners to operate on their own individual knowledge representation. Second, MTL relies on a priori strengths to select among the competing strategies. MBAC adapts the strategy selection process according to performance on the task. MTL's task adaptiveness resides only in the changing background knowledge according to information from the domain; whereas, MBAC adapts both the knowledge and the procedure for transforming the knowledge.

5.3 Adaptive Control

Adaptive control research related to the MBAC approach falls in the areas of model-reference adaptive control [Sastry and Bodson, 1989], system identification [Ljung, 1987], and intelligent control [Saridis, 1987]. Figure 5.8 shows a simple adaptive control loop. The update procedure uses feedback from the plant performance to adapt a model of the plant. This model provides the necessary control to reduce the error between the plant performance and desired performance. Model-reference adaptive control uses differences between model predictions and actual outcomes to update the model. System identification uses a variety of methods (e.g., regression) to identify the correct model of the plant. Intelligent control encompasses numerical as well as non-numeric methods for updating the model. For example, Michie and Chambers [1968] describe an adaptive control technique called Boxes for solving the pole-balancing problem. More recent work by Barto *et al.* [1983] also address the pole-balancing problem using neuron-like adaptive elements. Self [1990] describes the successful implementation of fuzzy adaptive control for the auto-focussing mechanism of a camera.

The main similarity of MBAC to adaptive control is the search for a model of the performance element (plant) based on feedback from the performance environment. Adaptive control analysis requires a mathematically expressible model in order to prove stability and convergence

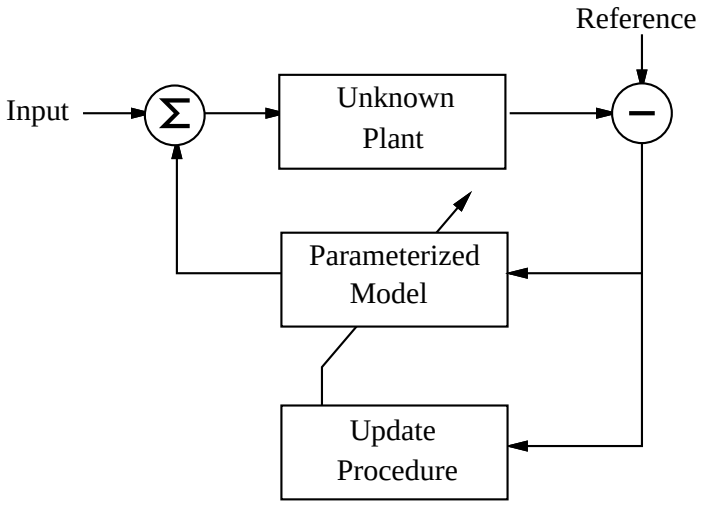

Figure 5.8: Adaptive control loop.

of the performance response. MBAC proposes a simple quadratic model related performance to the amount of learned knowledge for a variety of learning methods, performance elements (plants), and performance objectives. Although not a part of this investigation, results from adaptive control on proving system stability may be applicable to proving the stability of the MBAC approach when converging to desired performance objectives.

Chapter 6

Future Work

Several future directions exist for improving the analysis of the general utility problem and enhancing the MBAC approach. The current MBAC approach attempts to reduce dependence on knowledge of the performance environment in order to integrate several learning methods into a common framework. Figure 6.1 shows the spectrum of dependence on knowledge of the performance environment. MBAC resides at point B near the zero knowledge side of the spectrum. Systems described in previous chapters possess more knowledge of the performance environment and reside near point A of the spectrum. The optimal point along the spectrum resides somewhere in the middle, taking advantage of more knowledge while retaining a common framework for integrating multiple methods. The following future directions describe possible approaches for moving the MBAC approach further to the right along this spectrum.

6.1 Analysis of the General Utility Problem

The analysis of the general utility problem in Section 2.5 offers one approach to moving MBAC towards the use of more knowledge from the performance environment. This analysis attempts

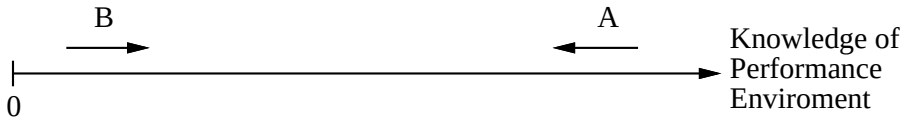

Figure 6.1: Spectrum measuring a system's knowledge of the performance environment. Systems near point A have more knowledge of the performance environment, whereas systems at point B have less knowledge of the performance environment. Arrows indicate future directions.

to derive a formal model for the relationship between performance and the amount of learned knowledge. As the analysis shows, these formal models may depend on the properties of the task domain. For example, the formal model for splitting methods depends on the number of training instances and the number of dimensions in instance space. The formal model for networks depends on the number of training instances. Empirical results using VBMS show that task domain properties are useful for selecting among multiple learning methods (see Section 5.2.1). This dependence on properties of the task domain is an attempt to estimate the complexity of the function to be learned. The formal models provide a more precise representation of the relationship between performance and the amount of learned knowledge.

Future work on the analysis of the general utility problem will continue to derive formal models for other learning methods. Although not yet at the stage of a mathematical model, the analysis of analytical learning methods in Section 2.5.2 identifies the components of the model. The next step is to derive a model based on these components and properties of the task domain (e.g., number of operators, complexity of operators, and size of the problem space). Formal models of other learning methods (e.g., analogy) will follow.

This research concentrates on learning methods and performance dimensions that combine to demonstrate the general utility problem trend. For example, plotting classification accuracy after each split of a splitting method results in the trend identified in Chapter 2. However, not all combinations of learning methods and performance dimensions yield a performance response following this trend. For example, the storage requirements of a neural network remain constant during the course of the network learning method, and hypothesis comprehensibility of a decision tree measured in terms of number of leaves increases monotonically with increasing numbers of splits. Although a second order curve (e.g., a parabola) can express these performance responses by setting the higher-order coefficients to zero, a model constrained to the proper order would be more appropriate for these cases. Formal analyses of these degenerate transformation/performance-dimension combinations will reveal the appropriate model.

As the formal models mature, MBAC will replace the parabolic model with the more precise formal model. MBAC's model of the performance response will no longer rely on empirical estimation. For learning methods still lacking a formal model, MBAC will retain the empirical model. Thus, MBAC provides a flexible architecture for the insertion of alternative formal

models; however, the models should retain enough constraints to enforce the general utility problem trend. The formal models derived in Section 2.5 follow this trend.

As with the formal models in Section 2.5, the new formal models will also depend on a measure of knowledge specific to the learning method. Future work will attempt to further unify the models by extending the measure to a more generic definition of knowledge. As alluded to by the requirement of a general to specific ordering of knowledge transformations, this new definition of knowledge will derive from an expression relating changes in knowledge to changes in knowledge generality. For example, additional splits in a splitting method increase the specificity of the learned hypothesis. The splits increase the degree of the function over instance space. Expressing the transformations in terms of their effects on hypothesis specificity will retain the ability to use formal models in place of empirical models and to integrate multiple learning methods within one framework based on one definition of knowledge.

Although a more precise definition for the amount of learned knowledge will move the MBAC approach further to the right in Figure 6.1, a much larger move towards the incorporation of performance-environment knowledge involves the determination of *which* knowledge affects performance. For example, the addition of one macro-operator by an analytical learning method may have a much greater effect on performance than another. Furthermore, the effect depends on the knowledge already learned by the method. The MetaLEX system (see Section 5.1.1) attempts to identify the knowledge responsible for a performance degradation and transforms this knowledge accordingly. The ability to identify individual pieces of knowledge directly responsible for changes in performance requires considerable knowledge of the performance environment. Extending MBAC to incorporate such knowledge is contradictory to MBAC's goal of unifying multiple learning paradigms. Each paradigm would require meta-knowledge of how each type of learnable knowledge can affect performance.

The general-to-specific constraint on the order of knowledge transformations addresses the issue of *which* knowledge to learn. The analysis in Section 2.5 requires the knowledge transformations to be ordered such that the learned hypothesis becomes more specialized with increasing numbers of transformations. Therefore, the constraint recommends that specific knowledge is responsible for degrading performance. Assuming the order of transformations satisfies this constraint, the issue of *which* knowledge is at fault reduces to determining the knowledge learned most recently, and controlling performance reduces to controlling the *amount* of learned knowl-

edge. Future work on more precise definitions of knowledge will rely on this ordering constraint to avoid incorporating considerable knowledge from the performance environment and to retain a unifying framework for multiple learning paradigms.

6.2 Model-Based Adaptive Control

Until formal models of learning methods are available, MBAC offers a useful empirical approach for avoiding the general utility problem. Improvements to the MBAC approach will utilize techniques from other systems described previously. These improvements move the MBAC approach to the right in Figure 6.1 toward increased utilization of performance environment knowledge.

One result from the experimentation in Chapter 4 is the sensitivity of the parabola estimation procedure to variations in the performance response. Although task transfer alleviated some of this sensitivity by forcing additional sampling, this sensitivity is still a problem. Other approaches address the sensitivity issue by maintaining more general models of the performance response. For example, MetaLEX maintains qualitative models describing the effects of transformations on performance (see Section 5.1.1). VBMS maintains a piecewise constant function as the model of the performance response and uses PLS1 to adapt instances of the model (see Section 5.2.1). AIMS has the ability to fit a variety of curves to the performance response (see Section 5.2.2). Future extensions to MBAC's modeling capabilities will migrate towards the more general models used by VBMS and AIMS. However, the migration will not be complete in order to retain the constraints identified in the performance response trend of Chapter 2. Approaches to this controlled migration include more robust curve-fitting techniques [Press *et al.*, 1986] and constraints on the more general modeling techniques exemplified in VBMS and AIMS.

One advantage of AIMS over MBAC is the ability to trade off multiple performance objectives. Section 3.5 addresses this issue, but the implementation of MBAC in Chapter 4 does not include this capability. Incorporation of multiple performance objectives into MBAC will require modifications to the transformation selection procedure (see Section 4.4.1). The main modification will add the ability to handle cases where two or more transformation recommendations achieve different Pareto optimal (non-dominated) points. The optimizer in AIMS

finds these Pareto optimal recommendations. Incorporation of this technique into MBAC will improve the transformation selection procedure when deciding among competing performance objectives.

Chapter 4 uses only empirical learning methods to evaluate the MBAC approach. A more complete evaluation requires application of the MBAC approach to other learning paradigms. This evaluation will not only confirm the usefulness of MBAC for these methods, but also provide the ability to compare different paradigms. For example, an alternative to an analytical learning method may be to use a structural induction method on the solutions to the training problems. Furthermore, one task may benefit from the application of more than one transformation. For example, MBAC might choose an empirical learning method to prove an intermediate fact needed in the solution of a problem using an analytical learning method. Evaluation of the MBAC approach on these additional learning methods requires further experimentation.

Chapter 7

Conclusions

The general utility problem in machine learning is the generation of low utility knowledge due to the uncontrolled application of machine learning methods. Uncontrolled selection of a learning method for a given learning task may generate low utility knowledge, because the selected method is inappropriate for the task. Uncontrolled execution of the selected learning method may generate low utility knowledge, because the method overfits the training data. Successful application of learning methods requires the selection of an appropriate learning method and the determination of the appropriate amount of knowledge to be generated by the method. Model-based adaptive control (MBAC) addresses these two control dimensions of the general utility problem by using a model of the performance response trend common among several different learning methods. This thesis investigates the MBAC approach to the general utility problem. Section 7.1 summarizes the results of this investigation, and Section 7.2 enumerates the contributions of the research.

7.1 Summary

The investigation of the general utility problem in machine learning begins in Chapter 2 by demonstrating the existence of the problem in several different learning paradigms. Chapter 2 introduces the performance response curve as a tool for observing the general utility problem. The performance response curve plots performance during the execution of the learning method. Plotting the performance response of several learning methods over several domains reveals a general trend in the curve as depicted in Figure 2.1. The performance response initially

increases to a single peak and then decreases at a lower rate. The fact that the performance decreases from the peak before termination of the learning method indicates the existence of the general utility problem. In addition, the peak of the performance response is higher than the performance achieved by popular methods for alleviating the general utility problem. Therefore, convergence to the peak of the response curve would improve the performance of the learning method. Section 2.5 formally analyzes several learning methods and derives formal models relating performance to the amount of learned knowledge. These formal models confirm the general utility problem trend of Figure 2.1.

The common trend in the performance response curves of several learning paradigms indicates that a model of the trend may be sufficient to control the amount of learned knowledge and avoid the general utility problem. Chapter 3 introduces the model-based adaptive control (MBAC) approach based on this observation. MBAC utilizes a parameterized curve to model the performance response. The curve adapts the parameters according to data points sampled from actual response curve data. With a model of the performance response, MBAC can predict the amount of learned knowledge necessary to achieve the performance objectives. By maintaining models for several learning methods, MBAC can select the most appropriate method for the learning task based on the models' predictions of achievable performance. Thus, MBAC combats the general utility problem by modeling the performance response and using the model to control the selection of learning methods and the generation of low utility knowledge.

In order to determine the effectiveness of the MBAC approach, Chapter 4 evaluates several components of MBAC. The success of MBAC depends on the predictive accuracy of the model. The first experiment shows that the parabolic model is superior to the rote and nearest-neighbor models. Experiment 1 also shows that the peak of the parabola corresponds closely to the peak of the true performance response. The success of the MBAC approach also depends on the ability to measure the certainty of the model. Experiment 2 compares three model certainty estimates according to their correlation to the ordering of learning methods from best to worst. Results indicate that the standard deviation of the model is an accurate estimate of model certainty. MBAC's success also depends on the accuracy of the models' predictions. The third experiment compares the predicted performance for some number of knowledge transformations to the actual performance obtained by performing the recommended number of transformations. Results show that the models closely predict the actual performance.

Experiments 4 and 5 evaluate the MBAC approach during the initial phase of adapting the model based on samples from the performance response. Experiment 4 illustrates MBAC's progress during this phase. Results indicate that MBAC adapts to the performance response, but can falter when the performance response contains local peaks. Experiment 5 shows how task transfer can alleviate some of this sensitivity to local peaks by transferring knowledge from other tasks.

MBAC exploits the general utility problem trend to maintain the utility of learned knowledge and select appropriate learning methods. The experimental results confirm the effectiveness of the MBAC approach over several learning methods and domains.

7.2 Contributions

The investigation of the general utility problem and the MBAC approach provides several contributions to research in machine learning. First, the realization that the utility problem occurs in several learning paradigms helps to unify machine learning methods. The unifying idea is the search for a concept in generalization space (the space of possible hypotheses ordered according to generality, similar to the version space). As in empirical learning, analytical learning attempts to find a concept (set of control rules or macro-operators) that maximizes performance. The concept must be at the correct level of generality that improves performance on a majority of the examples, but does not degrade performance by attending to lower-probability examples. The knowledge learned from the lower-probability examples will degrade performance. Learning methods that attempt to maximize performance via generalization risk suffer from the general utility problem.

The second contribution of this work is the observation that the performance response view of the general utility problem retains a common shape over several learning methods and task domains. Perception of the performance response trend requires new perspectives on how to learn. First, learning should proceed in smaller increments to allow the integration of feedback from the performance environment. The performance of simpler hypotheses considered during the course of current learning methods typically exceeds the performance of the final hypothesis. Second, if the small learning increments are taken in a specific order along the dimension of hypothesis generality, then control of the amount of learned knowledge is sufficient to control

the performance. Finally, the performance objectives of the learning task should be separate from the learning method. Explicit performance objectives allow more control over the learning and more flexibility in adapting to dynamic performance environments.

This work introduces and evaluates the MBAC approach. The performance response trend permits a single model for the relationship between performance and the amount of learned knowledge. Maintaining such a model for each combination of task domain, knowledge transformation and performance dimension, MBAC selects appropriate transformations and avoids generation of low utility knowledge. Furthermore, since the model is adaptive, MBAC can adapt the knowledge to changes in the performance environment. Experimental results confirm the applicability of MBAC to the general utility problem. Analysis of related work indicates that the stronger model of performance is the source of MBAC's more accurate control of learning.

Although the amount of learned knowledge is a course measure for controlling a learning method, the measure is useful in several situations where a more refined measure is unavailable. A system designed to learn in a variety of previously unknown domains will have little or no knowledge of the domain with which to control the learning. When the system attempts to control knowledge in an unknown performance environment (e.g., performance element is a black box), transformations to the knowledge have unknown effects on performance. Controlling the amount of learned knowledge helps to insure that the knowledge acquired by the system is within reasonable constraints. These constraints derive from the general utility problem trend and prevent performance degradation due to excessive amounts of learned knowledge.

Finally, this work contributes a preliminary formal analysis of the general utility problem in several learning methods. The analysis confirms the trend identified in Chapter 2. Refinement of the model-based adaptive control approach according to the enhancements described in Chapter 6, and incorporation of more formal models of the performance response will evolve the approach into a general methodology for maintaining the utility of learned knowledge. At the heart of the methodology will be the model of the performance response whose common shape serves to unify multiple learning paradigms under one framework.

APPENDIX A

Domains

A.1 Empirical Learning Domains

Five domains are used to test the empirical learning methods: DNF2, Breast Cancer, Flag, Flare, and Voting. The first domain, DNF2, is defined by a DNF concept over forty binary-valued features that appears in [Pagallo and Haussler, 1990]. In all trials, 1000 randomly-chosen examples were used for training and 500 for testing. The concept is reproduced below:

$$\begin{aligned}\text{DNF2: } & x_1x_3x_{14}x_{19}x_{26}x_{35}x_{36} + x_8x_{15}x_{31}x_{37} + x_5x_{10}x_{14}x_{27}x_{29} + \\ & x_{18}x_{20}x_{30}x_{36} + x_2x_3x_9x_{19}x_{24} + x_{24}x_{25}x_{27}x_{36}x_{37} + \\ & x_6x_7x_{14}x_{25}x_{26}x_{31}x_{34} + x_1x_6x_{22}x_{30}\end{aligned}$$

The remaining four domains come from the UC Irvine database. For each domain, missing feature values were filled in probabilistically according to the distribution of values in other examples having the same classification. In all trials, two-thirds of the entire dataset were randomly chosen without replacement to comprise the training set. The remaining one-third of the examples were used for testing. In the case of reduced-error pruning, the training, pruning and testing sets were chosen similarly in the ratios one-half, one-fourth, and one-fourth, respectively.

The Breast Cancer database was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia due to M. Zwitter and M. Soklic. The database contains 286 examples over nine features and a binary-valued class that indicates whether or not breast cancer will recur.

The Flag database was collected from the *Collins Gem Guide to Flags* by R. S. Forsyth. The database contains 194 examples over 30 features of which 10 were removed due to their lack of relation to the flag descriptions. The religion feature was combined into a binary-valued class feature corresponding to eastern and western religions.

The Flare database contains 1066 examples over 10 features describing regions of the sun, and a binary-valued class feature corresponding to the production of one or more solar flares in that region in the next 24 hours.

The Voting database was collected from the Congressional Quarterly Almanac and contributed by J. Schlimmer. The database contains 435 examples over 16 features describing voting records, and a binary-valued class feature corresponding to the political party.

A.2 Analytical Learning Domains

Experimentation with the analytical learning method uses two domains: blocks and robot. The blocks domain consists of four operators for moving blocks in the blocks-world. The robot domain consists of eight operators using a robot to move boxes within a layout of connected rooms. The following sections further describe the analytical task domains and list the operators.

A.2.1 Blocks Domain

The blocks domain consists of the four operators shown below for stacking and unstacking blocks in the blocks world. Given the number of blocks n, the problem generator returns a randomly-selected initial state and goal state. The states are generated by placing the n blocks on the table in n columns. The state generator randomly chooses a column $(1 \ldots n)$ for each block. If two blocks have the same column, the blocks are stacked. For the experiments using the blocks domain, $n = 3$, the number of training examples is 100, and the number of testing examples is 50.

```
(Operator (pickup ?x)
    :conditions ((clear ?x) (ontable ?x) (handempty))
    :delete-list ((ontable ?x) (clear ?x) (handempty))
    :add-list ((holding ?x)))

(Operator (putdown ?x)
    :conditions ((holding ?x))
```

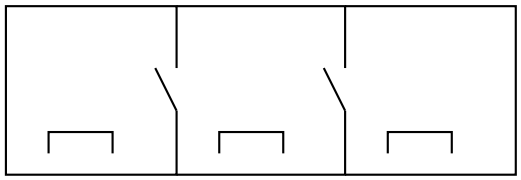

Figure A.1: Three-room configuration used to generate initial and goal states for the robot domain.

```
    :delete-list ((holding ?x))
    :add-list ((clear ?x) (handempty) (ontable ?x)))

(Operator (stack ?x ?y)
    :conditions ((clear ?y) (holding ?x))
    :delete-list ((holding ?x) (clear ?y))
    :add-list ((handempty) (clear ?x) (on ?x ?y)))

(Operator (unstack ?x ?y)
    :conditions ((on ?x ?y) (clear ?x) (handempty))
    :delete-list ((on ?x ?y) (clear ?x) (handempty))
    :add-list ((holding ?x) (clear ?y)))
```

A.2.2 Robot Domain

The robot domain consists of the eight operators shown below using a robot to move boxes within a layout of connected rooms. Figure A.1 shows the three-room layout used in the experimentation. Each room has a table and a door to the adjoining room. Given the number of boxes n, the problem generator returns a randomly-selected initial states and goal state. The states are generated by randomly placing the n boxes and the robot into the three rooms. For the experiments using the robot domain, $n = 2$, the number of training examples is 25, and the number of testing examples is 25.

```
(Operator (goto-door-from-table ?room ?door)
    :conditions ((in-room ?room)
                 (next-to TABLE)
                 (door-to-room ?door ?room))
    :delete-list ((next-to TABLE))
    :add-list ((next-to ?door)))

(Operator (goto-door-from-door ?room ?door1 ?door2)
    :conditions ((in-room ?room)
                 (next-to ?door1)
                 (door-to-room ?door2 ?room))
```

```
    :delete-list ((next-to ?door1))
    :add-list ((next-to ?door2)))

(Operator (open-door ?door)
    :conditions ((next-to ?door) (closed ?door))
    :delete-list ((closed ?door))
    :add-list ((open ?door)))

(Operator (close-door ?door)
    :conditions ((next-to ?door) (open ?door))
    :delete-list ((open ?door))
    :add-list ((closed ?door)))

(Operator (go-thru-door ?door ?room1 ?room2)
    :conditions ((in-room ?room1)
                 (next-to ?door)
                 (door-to-room ?door ?room1)
                 (door-to-room ?door ?room2)
                 (open ?door))
    :delete-list ((in-room ?room1))
    :add-list ((in-room ?room2)))

(Operator (goto-table-from-door ?room ?door)
    :conditions ((in-room ?room)
                 (next-to ?door)
                 (door-to-room ?door ?room))
    :delete-list ((next-to ?door))
    :add-list ((next-to TABLE)))

(Operator (pickup-box ?box ?room)
    :conditions ((on-table ?box ?room)
                 (in-room ?room)
                 (next-to TABLE)
                 (arms-empty))
    :delete-list ((on-table ?box ?room) (arms-empty))
    :add-list ((holding ?box)))

(Operator (putdown-box ?box ?room)
    :conditions ((in-room ?room) (next-to TABLE) (holding ?box))
    :delete-list ((holding ?box))
    :add-list ((on-table ?box ?room) (arms-empty)))
```

References

[Barron, 1984] A. R. Barron. Predicted squared error: A criterion for automatic model selection. In S. J. Farlow, editor, *Self-Organizing Methods in Modeling*, chapter 4, pages 87–103. Marcel Dekker, 1984.

[Barto *et al.*, 1983] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):834–846, 1983.

[Bhansali and Harandi, 1991] S. Bhansali and M. T. Harandi. Synthesizing unix shell scripts using derivational analogy: An empirical assessment. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 521–526, 1991.

[Breiman *et al.*, 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[Buchanan *et al.*, 1978] B. G. Buchanan, T. M. Mitchell, R. G. Smith, and C. R. Johnson. Models of learning systems. In J. Belzer, editor, *Encyclopedia of Computer Science and Technology Vol. 11*. Marcel Dekker, Inc., 1978.

[Carbonell, 1986] J. G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol II*, chapter 14, pages 371–392. Morgan Kaufmann Publishers, 1986.

[Carlson *et al.*, 1990] B. Carlson, J. Weinberg, and D. Fisher. Search control, utility, and concept induction. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 85–92, 1990.

[Cestnik *et al.*, 1987] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge-elicitation tool for sophisticated users. In *Proceedings of the Second European Working Session on Learning*, pages 31–45, 1987.

[Clark and Niblett, 1989] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–284, 1989.

[Cohen, 1990] W. W. Cohen. Learning approximate control rules of high utility. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 268–276, 1990.

[DeJong and Mooney, 1986] G. F. DeJong and R. J. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, April 1986.

[Dietterich and Buchanan, 1983] T. G. Dietterich and B. G. Buchanan. The role of the critic in learning systems. In E. W. Rissland, M. Arbib, and O. Selfridge, editors, *Adaptive Control of Ill-Defined Systems*. Plenum, 1983.

[Dietterich *et al.*, 1982] T. G. Dietterich, B. London, K. Clarkson, and G. Dromey. Learning and inductive inference. In P. R. Cohen and E. A. Feigenbaum, editors, *The Handbook of Artificial Intelligence, Volume III*, chapter XIV, pages 323–512. HeurisTech Press and William Kaufmann, 1982.

[Etzioni, 1988] O. Etzioni. Hypothesis filtering: A practical approach to reliable learning. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 416–429, 1988.

[Fikes *et al.*, 1972] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 4(3):189–208, 1972.

[Freund, 1988] J. E. Freund. *Modern Elementary Statistics*. Prentice-Hall, seventh edition, 1988.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[Gratch and DeJong, 1991] J. Gratch and G. DeJong. A hybrid approach to guaranteed effective control strategies. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 509–513, 1991.

[Holder, 1990] L. B. Holder. The general utility problem in machine learning. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 402–410, 1990.

[Holte *et al.*, 1989] R. C. Holte, L. E. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, 1989.

[Karnin, 1990] E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[Keller, 1987a] R. M. Keller. Concept learning in context. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 91–102, 1987.

[Keller, 1987b] R. M. Keller. *The Role of Contextual Knowledge in Learning Concepts to Improve Performance*. PhD thesis, Department of Computer Science, Rutgers University, January 1987.

[Keller, 1988] R. M. Keller. Defining operationality for explanation-based learning. *Artificial Intelligence*, 35(2):227–241, June 1988.

[Ljung, 1987] L. Ljung. *System Identification: Theory for the User*. Prentice-Hall, 1987.

[Michalski *et al.*, 1986] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application in three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1047, 1986.

[Michalski, 1989] R. S. Michalski. How to learn imprecise concepts: A method based on two-tiered representation and the AQ15 program. In Y. Kodratoff and R. S. Michalski, editors, *Machine Learning: An Artificial Intelligence Approach, Vol III.* Morgan Kaufmann Publishers, 1989.

[Michie and Chambers, 1968] D. Michie and R. A. Chambers. Boxes: An experiment in adaptive control. *Machine Intelligence*, 2:137–152, 1968.

[Mingers, 1989a] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, November 1989.

[Mingers, 1989b] J. Mingers. An empirical comparison of selection measures for decision tree induction. *Machine Learning*, 3(4):319–342, March 1989.

[Minton, 1985] S. Minton. Selectively generalizing plans for problem-solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 596–599, 1985.

[Minton, 1988a] S. Minton. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer Academic Publishers, 1988.

[Minton, 1988b] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 564–569, 1988.

[Minton, 1990] S. Minton. Issues in the design of operator composition systems. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 304–312, 1990.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, January 1986.

[Mooney and Bennett, 1986] R. J. Mooney and S. W. Bennett. A domain independent explanation-based generalizer. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 551–555, 1986.

[Mooney, 1989] R. J. Mooney. The effect of rule use on the utility of explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 725–730, 1989.

[Mozer and Smolensky, 1989] M. C. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

[O'Rorke, 1987] P. V. O'Rorke. LT revisited: Experimental results of applying explanation-based learning to the logic of principia mathematica. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 148–159, 1987.

[Pagallo and Haussler, 1990] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–100, 1990.

[Press *et al.*, 1986] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[Quinlan and Rivest, 1989] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1987] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Rao, 1984] S. S. Rao. *Optimization: Theory and Applications*. Halsted Press, second edition, 1984.

[Rendell *et al.*, 1987a] L. Rendell, R. Seshu, and D. Tcheng. Layered concept learning and dynamically-variable bias management. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 308–314, 1987.

[Rendell *et al.*, 1987b] L. Rendell, R. Seshu, and D. Tcheng. More robust concept learning using dynamically-variable bias. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 66–78, 1987.

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

[Rissanen, 1989] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Publishing Company, 1989.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing, Volume 1*, chapter 8, pages 318–362. MIT Press, 1986.

[Saridis, 1987] G. N. Saridis. Knowledge implementation: Structures of intelligent control systems. In *IEEE Symposium on Intelligent Control*, pages 9–17, 1987.

[Sastry and Bodson, 1989] S. Sastry and M. Bodson. *Adaptive Control: Stability, Convergence and Robustness*. Prentice-Hall, 1989.

[Self, 1990] K. Self. Designing with fuzzy logic. *IEEE Spectrum*, 27(11), 1990.

[Shavlik, 1988] J. W. Shavlik. *Generalizing the Structure of Explanations in Explanation-Based Learning*. PhD thesis, Department of Computer Science, University of Illinois, January 1988.

[Tambe and Newell, 1988] M. Tambe and A. Newell. Some chunks are expensive. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 451–458, 1988.

[Tambe and Rosenbloom, 1989] M. Tambe and P. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 731–737, 1989.

[Tcheng *et al.*, 1989] D. K. Tcheng, B. L. Lambert, S. C. Lu, and L. A. Rendell. Building robust learning systems by combining induction and optimization. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 806–812, 1989.

[Tcheng *et al.*, 1991] D. K. Tcheng, B. L. Lambert, S. C. Lu, and L. A. Rendell. AIMS: An adaptive interactive modeling system for supporting engineering decision making. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 645–649, 1991.

[Tecuci and Michalski, 1991] G. D. Tecuci and R. S. Michalski. A method for multistrategy task-adaptive learning based on plausible justifications. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 549–553, 1991.

VITA

Lawrence Bruce Holder, Jr. was born on October 2, 1964 in Granite City, Illinois. He entered the University of Illinois at Urbana–Champaign in August 1982 and graduated with honors in May 1986 with a Bachelor of Science degree in computer engineering. He then continued into graduate school at the University of Illinois, where he worked with Dr. Robert Stepp in the area of machine learning. In May 1988 he earned a Master of Science degree in computer science. He continued his research in machine learning under Dr. Larry Rendell and earned the Doctor of Philosophy degree in computer science from the University of Illinois in October 1991. Currently, he is an assistant professor in the Computer Science Engineering department at the University of Texas at Arlington.