EMPIRICALLY-BASED SELF-DIAGNOSIS AND REPAIR OF DOMAIN KNOWLEDGE

A Thesis
Presented to
The Academic Faculty

by

Joshua K. Jones

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
May 2010

EMPIRICALLY-BASED SELF-DIAGNOSIS AND REPAIR OF DOMAIN KNOWLEDGE

Approved by:

Professor Ashok Goel, Advisor
College of Computing
Georgia Institute of Technology

Professor Thomas Dietterich
School of Electrical Engineering and Computer Science
Oregon State University

Professor Alexander Gray
College of Computing
Georgia Institute of Technology

Professor Charles Isbell
College of Computing
Georgia Institute of Technology

Professor Ashwin Ram
College of Computing
Georgia Institute of Technology

Date Approved: 7 December 2009

To those that have cared about me,
for providing support in difficult times.

To those who have not believed in me,
for providing motivation.

And above all, to my friends and family,
for mostly sticking to the former.

ACKNOWLEDGEMENTS

I must primarily thank Dr. Ashok Goel, my advisor, without whose guidance, both intellectual and personal, none of this would have been possible. Each of the members of my committee were important in helping to shape the research detailed in this document. I particulary want to acknowledge Dr. Charles Isbell for many valuable discussions along the way, and for welcoming me as a sometime member of his group. I also must specifically acknowledge Dr. Thomas Dietterich, whose detailed and thoughtful feedback has had a substantial impact on this dissertation. Dr. Alexander Gray and Dr. Ashwin Ram each provided alternative perspectives on the research and asked deep questions that improved the quality of the work without a doubt. I also had many valuable discussions with students at Georgia Tech, particularly past and present members of the Design & Intelligence Laboratory and the The Laboratory for Interactive Artificial Intelligence. In particular, Michael Helms, Michael Holmes, Maithilee Kunda and Brian Sherwell have been very important to me during my journey through the Ph.D. process.

Though it is not practical to do so by name, I also wish to thank all of my friends both in Atlanta and elsewhere, for making this both possible and meaningful. I also want to thank my family – Kenneth, Christine and Amanda.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this work, I view incremental experiential learning in intelligent software agents as progressive agent self-adaptation. When an agent produces an incorrect behavior, then it may reflect on, and thus diagnose and repair, the reasoning and knowledge that produced the incorrect behavior. In particular, I focus on the self-diagnosis and self-repair of an agent's domain knowledge. The implementation of systems with the capability to self-diagnose and self-repair involves building both reasoning processes capable of such learning and knowledge representations capable of supporting those reasoning processes. The core issue my dissertation addresses is: what kind of metaknowledge (knowledge about knowledge) may enable the agent to diagnose faults in its domain knowledge? In providing a solution to this issue, the central contribution of this research is a theory of the kind of metaknowledge that enables a system to reason about and adapt its conceptual knowledge. For this purpose, I propose a representation that explicitly encodes metaknowledge in the form of procedures called *Empirical Verification Procedures* (EVPs). In the proposed knowledge representation, an EVP is associated with each concept within the agent's domain knowledge. Each EVP explicitly semantically grounds the associated concept in the agent's perception, and can thus be used as a test to determine the validity of knowledge of that concept during diagnosis.

I present the formal and empirical evaluation of a system, Augur, that makes use of EVP metaknowledge to adapt its own domain knowledge in the context of a particular subclass of classification problem that I call *compositional classification*, in which the overall classification task can be broken into a hierarchically organized set of subtasks. I hypothesize that EVP metaknowledge will enable a system to automatically adapt its knowledge in two ways: first, by adjusting the ways that inputs are categorized by a concept, in accordance with semantics fixed by an associated EVP; and second, by adjusting the semantics of concepts themselves when they fail to contribute appropriately to system goals. The latter adaptation is realized by altering the EVP associated with the concept in question. I further hypothesize that the semantic grounding of domain concepts in perception through the use of EVPs will increase the generalization power of a learner that operates over those concepts, and thus make learning more efficient. Beyond the support of these hypotheses, I also present results pertinent to the understanding of learning in compositional classification settings using structured knowledge representations.

CHAPTER I

INTRODUCTION

It is generally agreed in AI that the capability of metareasoning, reasoning about reasoning, is essential for achieving human-level intelligence [10] [75] [74] [103]. A canonical metareasoning architecture is depicted in Figure 1. Metareasoning systems extend the basic agent view of a software system, where the program receives percepts from and acts within an environment (called the *object* level in Figure 1), to include a reflective layer that *monitors* the agent processing and exerts *control* over it, e.g. by altering reasoning strategies being used at the object level if it becomes apparent that progress is not being made. Past AI research has shown that metareasoning is useful for control of reasoning [24] [104] [45] [44] [91], bounding of computations [48] [95] [47], self-explanation [42] [41] [22] [21] [39], method selection [80] [90], impasse resolution [63], revision of conclusions [27], revision of reasoning processes [108] [65] [86] [9] [102] [104] [105] [22] [87] [85] [85] [38] [99] [59], refinement of indices [33] [80], and guiding of reinforcement learning [121] [120] [122] [5] [99]. Cox [20] and Anderson & Oates [4] provide a useful review of AI research on metareasoning.

AI research on metareasoning for agent self-adaptation has generally focused on modifying the agent's reasoning processes. The need for self-adaptation of course arises because intelligent agents typically operate in dynamic task environments. It is useful to make a few distinctions here. Firstly, adaptations to an agent can be retrospective (i.e., when the agent fails to achieve a goal in its given environment, [9] [109] [110] [102] [104] [105] [22] [75] [83] [87] [38] [99] [59] [65] [128]), or proactive (i.e., when the agent is asked to operate in a new task environment, e.g., [86], [87] [85] [59]). Secondly, adaptations can be either to the deliberative element in the agent

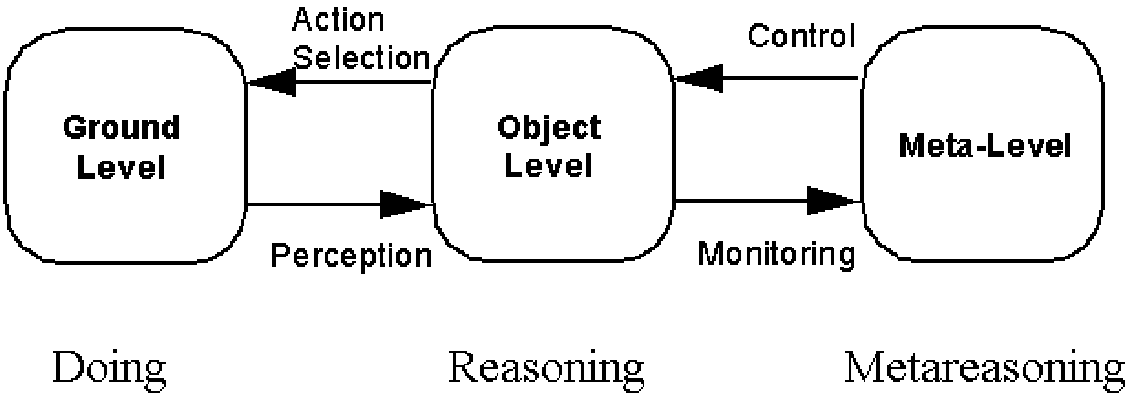


Figure 1: Canonical metareasoning architecture, adapted from [19].

architecture [9] [128] [102] [104] [105] [22] [87] [85] [85] [38] [99] [59] [108] [65] [86], or the reactive element [111], or both. Thirdly, adaptations to the deliberative element may be modifications to its reasoning process (i.e., to its task structure, selection of methods, or control of reasoning, e.g., [9] [108] [86]), or to its domain knowledge (i.e., the content, representation and organization of its knowledge, [65] [128] [80] [53] [54] [55] [57] [56] [58]), or both.

A core and longstanding problem in self-adaptation is that of credit (or blame) assignment [98] [75]. It is useful to distinguish between two kinds of credit assignment problems: temporal and structural. In temporal credit assignment, given a sequence of many actions by an agent that leads to a failure, the task is to identify the actions(s) responsible for the failure. Reinforcement learning is one method for addressing the temporal credit assignment problem [114]. In structural credit assignment, or diagnosis, given an agent composed of many knowledge and reasoning elements that fails to achieve a goal, the task is to identify the element(s) responsible for the failure. Metareasoning for self-adaptation typically addresses the problem of structural credit assignment [121] [122], though it can also be used to guide reinforcement learning [120]

[5]. It is worth noting the close relationship between agent self-adaptation and agent learning: the use of metareasoning for self-adaptation views learning as a deliberative, knowledge-based process of self-diagnosis and self-repair.

As indicated above, existing AI research on metareasoning for self-adaption has generally focused on modifying the agent's reasoning processes. When required to reason about and adapt its own reasoning process, an agent may in many cases benefit from access to an explicit representation of that process (often called a *model*), including some knowledge about what the various reasoning elements represented in the model are meant to achieve. Knowing something about the desired results of the execution of a reasoning element is important because this knowledge allows the reasoner to identify unexpected behavior and thus likely malfunctions. Without any notion of how a thing is meant to behave, there is no reasonable basis to make judgements about its actual behavior. Notice that there is a *predictive* character to the knowledge used by the agent for self-diagnosis – expectations associated with reasoning elements will be either respected or violated in practice, forming the basis for the agent's self-diagnosis. When diagnosing problems in its reasoning process, an agent may in some cases localize a fault to a primitive task that is directly implemented by domain knowledge. In such a case, we would like the agent to be able to reflect upon and correct the impugned knowledge. A concrete example of this type of metareasoning scenario is given in the following subsection. In this dissertation, we describe work on this important problem, enabling metareasoning agents to reflect upon and modify the agent's domain knowledge. The central question addressed by this research is: what is the form of metaknowledge that will be useful to an agent in reasoning over and adapting its own knowledge? The overarching hypothesis adopted by this work is that, analogous to the predictive knowledge used in self-adaptation of reasoning process, *knowledge about domain knowledge (metaknowledge) should also be specified in the form of verifiable predictions.*

A major contribution of this work is in laying the groundwork for a theory of metareasoning-based adaptation of domain knowledge through a refinement of this general hypothesis. In refining the overarching hypothesis stated above, the next question that arises is: how can one explicitly operationalize the verifiable predictions implied by domain knowledge such that the agent that uses the knowledge can automatically check the correctness of that domain knowledge? The answer we propose is that each piece of an agent's knowledge should have associated with it procedures consisting of sequences of actions and observations in the environment that can be used to test the veracity of an associated piece of domain knowledge. We call these pieces of metaknowledge Empirical Verification Procedures (EVPs), and hypothesize that they are a form of metaknowledge that will enable a system to successfully extend self-diagnosis to include domain knowledge, as well as adapt knowledge that is identified as faulty. An interesting conceptual implication of this hypothesis is that domain knowledge with associated EVPs acquires its semantics through grounding in perception, because that knowledge will be considered correct only if it leads to accurate predictions about perception and modified to conform to that ideal of correctness otherwise.

Fixing the semantics of concepts in perception with EVPs also constrains the expressivity of those concepts, as they are free only to express knowledge consistent with the perceptual expectations to which they are tied. For this reason, we expect that concepts within an overall knowledge structure that are semantically "pinned" with EVPs will be learned more quickly (in terms of the number of examples required to learn the concept) than those that are not. Chapter 5 describes our empirical evaluation of this hypothesis.

An additional, substantial benefit of explicitly encoding an agent's conceptual semantics in such a procedure is that *the procedures themselves become first class objects that can be operated upon.* In this way, an agent can change the semantics of its own concepts automatically if there is reason to believe that an altered concept would better contribute to the solution of the overall classification problem. Experimental results with such adaptations are described later, in Chapter 6. Given that an agent has the capability to alter the sets of equivalence classes into which it abstracts perceived scenarios, a question is immediately raised: upon what basis should an agent decide to alter the semantics of one of its concepts? Or, to state the question positively, what makes a particular abstraction useful? The answer given by this research leads to a second, related but distinct hypothesis that we adopt with respect to knowledge, that *a concept's value ultimately stems from its ability to support action selection.* In practice, this means that when an agent sees evidence that one of its concepts is not adequately playing its role in terms of supporting the successful achievement of the agent's tasks, the agent should take steps to directly modify that concept's semantics by altering the associated EVP. We specifically hypothesize that the incorporation of such mechanisms will make an agent more successful at achieving its tasks when some of its concepts may have suboptimal semantics in terms of their ability to support those tasks. This hypothesis is tested empirically in Chapter 6.

In order to test the hypotheses enumerated in the preceding paragraphs, we must refine them still further within the context of a specific problem so that we arrive at an implementable level of detail. Since classification is a ubiquitous task in AI ([43] [15]), we have chosen to consider the problem of using metaknowledge for repairing classification knowledge when the classifier supplies an incorrect class label. More specifically, we consider the subclass of classification problems that can be decomposed into a hierarchical set of smaller classification problems; alternatively, problems in which features describing the world are progressively aggregated and abstracted into higher-level abstractions until a class label is produced at the root node. This subclass of classification problems is recognized as capturing a common pattern of classification (e.g., [37] [97]). In fact, this class of problems is so common that Chandrasekaran ([13] [14]) identified it as a Generic Task. We will call this classification task *compositional classification*, and the hierarchy of abstractions an *Abstraction Network*. In particular, we consider the problem of retrospective adaptation of the content of the intermediate abstractions in the Abstraction Network (and *not* its structure) when the classifier makes an incorrect classification.

Compositional classification is a particularly interesting domain in which to test the efficacy of EVP-based self-adaptation not only because it appears so commonly within AI systems, but also because of the somewhat indirect relationship between concepts within a classification hierarchy and the actions that are taken by an agent in pursuit of its goals. A top-level classification, which is directly produced by a non-compositional, monolithic classifier, and ultimately produced by any type of classifier, is typically tightly integrated with action selection in a complete agent architecture. Therefore, it is likely that process-level expectations about the action selected based upon a top-level classification will serve to verify that classification. That is, the key prediction an agent makes through such a classification is that a certain action will have a desired consequence. In contrast, intermediate classifications within a hierarchy have one major purpose – to support the production of a correct top-level classification (and ultimately, action selection). So, for these intermediate classifications, no individual verification will take place in the course of the agent's normal processing. It is for this reason that EVPs are most pertinent within *structured* knowledge representations. And indeed, the notion of deliberative self-diagnosis makes sense only within such representations.

Refining our overall hypotheses in the context of compositional classification means that intermediate abstractions in the Abstraction Network are chosen such that each abstraction corresponds to a prediction about percepts in the world, metaknowledge comes in the form of verification procedures (EVPs) associated with the abstractions, and metareasoning invokes the appropriate EVPs to perform structural credit assignment and then adapt the abstractions. The EVPs explicitly encode the grounding of intermediate abstractions in percepts from the environment, and will be modified when the agent sees evidence that the associated abstraction fails to support appropriate inference at the parent. This architecture for compositional classification is depicted in Figure 2. To support empirical evaluation of our theory within the domain of compositional classification, we have implemented a system, Augur, that makes use of EVPs for self-adaptation of compositional classification knowledge. In the remainder of this thesis we illustrate, formalize and evaluate the use of EVPs for self-adaptation of domain knowledge in Abstraction Networks, and present empirical results obtained by applying Augur in both synthetic and real domains.

These hypotheses, and the corresponding observation about the predictive nature of the knowledge used to adapt reasoning processes, suggest an elaboration of the canonical metareasoning architecture of Figure 1, depicted in Figure 3. In the view of metareasoning taken in this work, the meta-level detects errors in processing and/or knowledge at the object level based on violations of expectations expressed in terms of the environment. Thus, the meta-level needs to observe not only the object level, but also the ground level. Further, when problems are identified at the object level by this monitoring, the meta-level may need to cause the system to take some actions in the environment in order to gather more information needed to resolve the problems. For example, the meta-level may execute EVPs at intermediate nodes in a classification hierarchy to determine which pieces of knowledge are responsible for an observed top-level classification error. Finally, as shown in Figure 2, metaknowledge used by the meta-level process may be directly distributed over the object level knowledge structures rather than being strictly confined to separate representations of the meta-level – here, EVPs are encoded as part of an agent's hierarchical classification knowledge.

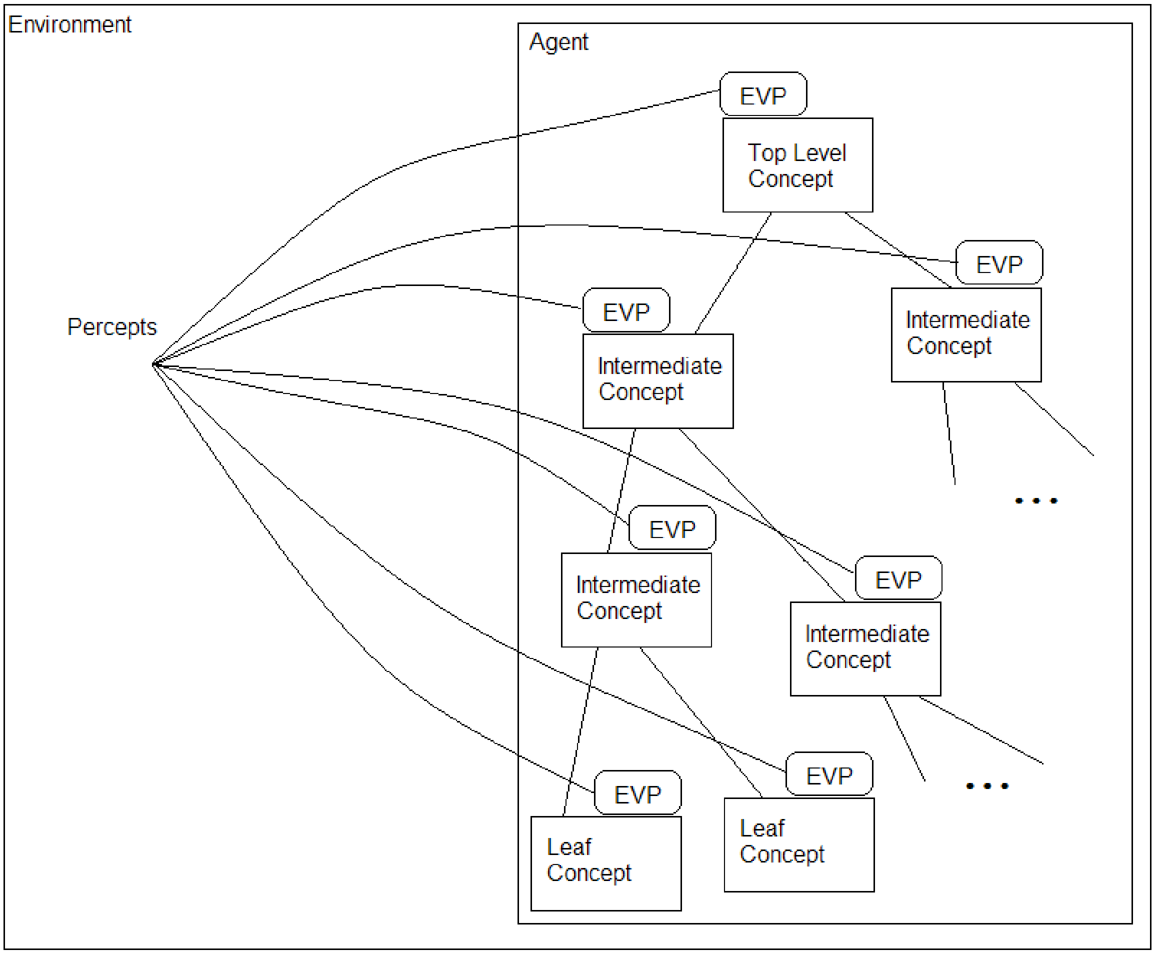


Figure 2: Hierarchical classification knowledge structure with Empirical Verification Procedures grounding concepts in perception.

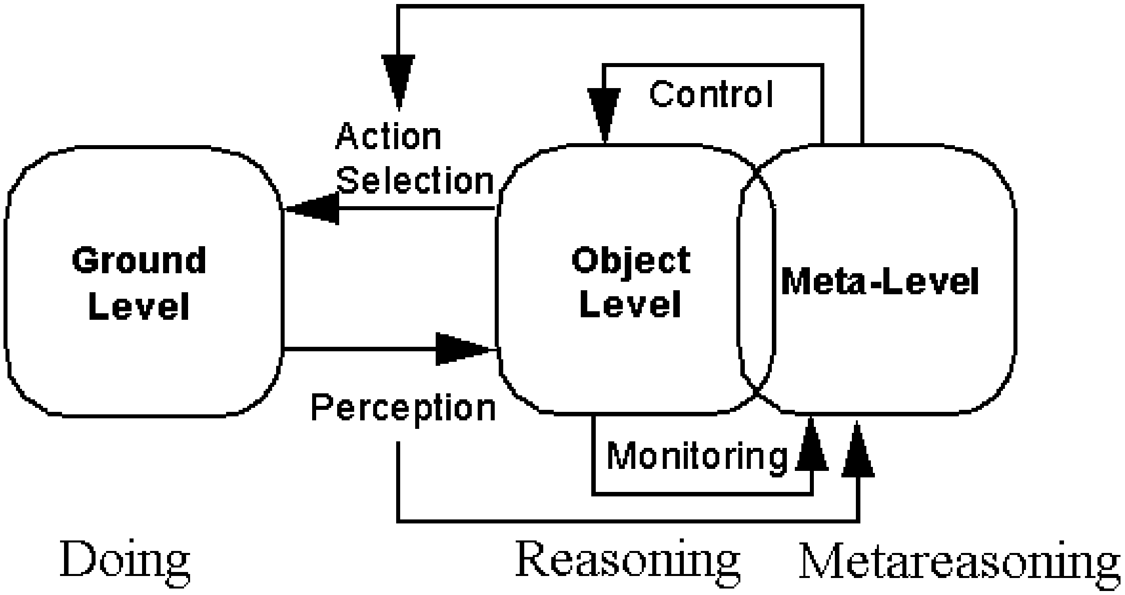


Figure 3: Elaborated metareasoning architecture.

1.1 Problem Domain

The general problem area in which we test the effectiveness of EVPs in enabling efficient diagnosis and adaptation of domain knowledge is classification. Here, we will consider a general classification problem to be one that requires the prediction of a class label, t, given some set of features (random variables), F, the values of which carry at least some information about the probable value of the class label. A problem instance is obtained by jointly sampling $F \cup \{t\}$, and providing the obtained values of the variables in F to the classification system. The system is considered to have correctly classified the example if it accurately produces the (hidden) sampled value of t, and incorrect otherwise. Here we are concerned with classification *learning*, where the classification system is not imbued a priori with complete knowledge of the function from features to most probable classification labels, but must instead infer this function from experience.

Given this description of the overarching classification problem, let us now step

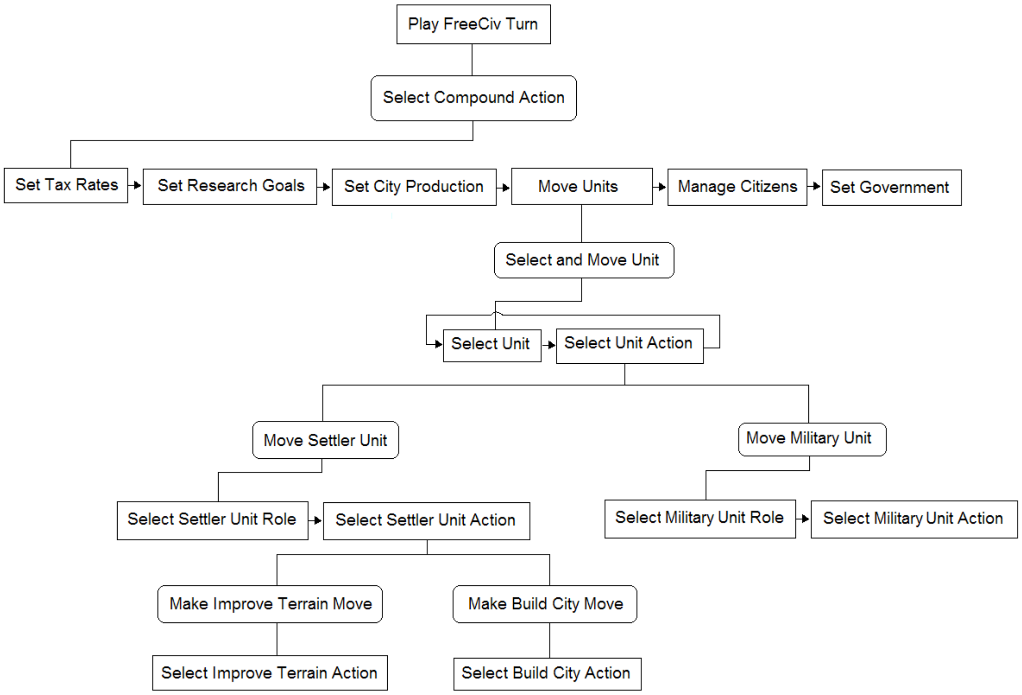


Figure 4: FreeCiv agent process model.

back and describe in detail a more specific class of learning problems, *compositional* classification tasks, a subclass of which is used to test the hypotheses of this research. We begin by providing an example of compositional classification.

1.1.1 Introductory Example

To make the problem concrete, we will present an example from the turn-based strategy game called FreeCiv (www.freeciv.org). Figure 4 depicts an example of a partially expanded process model for an agent that plays the game. This model is expressed in a teleological modeling language, Task-Method Knowledge Language (TMKL) [86]. There is more description of TMKL in Chapter 8, but for the purposes of this discussion it is sufficient to understand that Figure 4 can be understood as a task-subtask decomposition of the game playing agent's processing.

On each turn in a game of FreeCiv, the agent depicted in Figure 4 must select a compound action that consists of setting various parameters and moving units such as military units and “worker” units, called settlers, that can improve terrain or build new cities. Building new cities on the game map is a crucial action, as each city produces resources on subsequent turns that can then be used by the player to further advance their civilization. The quantity of resources produced by a city on each turn is based on various factors, including the terrain and special resources surrounding the city’s location on the map, and the skill with which the city’s operations are managed. The agent modeled in Figure 4 handles decisions about moving units to build cities in the subtask *Select Build City Action*. Consider what happens when meta-level monitoring detects that the game playing agent has made some error, perhaps failing in its overall goal of winning a game. At this point, a diagnostic procedure like that implemented in the metareasoning system REM [86] is engaged, and the agent reasons over its self-model of object level processing in order to localize the cause for failure. In some situations, this process of self-diagnosis may lead to the identification of some primitive task in the process model as a cause for failure. Primitive tasks are those that are directly achievable by applying some knowledge and/or taking some action in the world. Frequently, these primitive tasks may fundamentally be compositional classification tasks. In this work, we consider the self-diagnosis and self-repair problem that arises when the agent identifies a task such as the *Select Build City Action* primitive task as the cause of a failure, and the agent must reflect upon and correct its domain knowledge.

In the current example, when our agent selects the action for a unit that is to build a city, a crucial decision is whether the location on the game map currently occupied by the unit is suitable for the placement of the new city. We will judge the quality of a potential city location based upon the quantity of resources that we expect a city built in that location to produce over time. This decision is an example of a compositional classification task. Figure 5 illustrates a knowledge hierarchy for this task used by our FreeCiv game-playing agent.

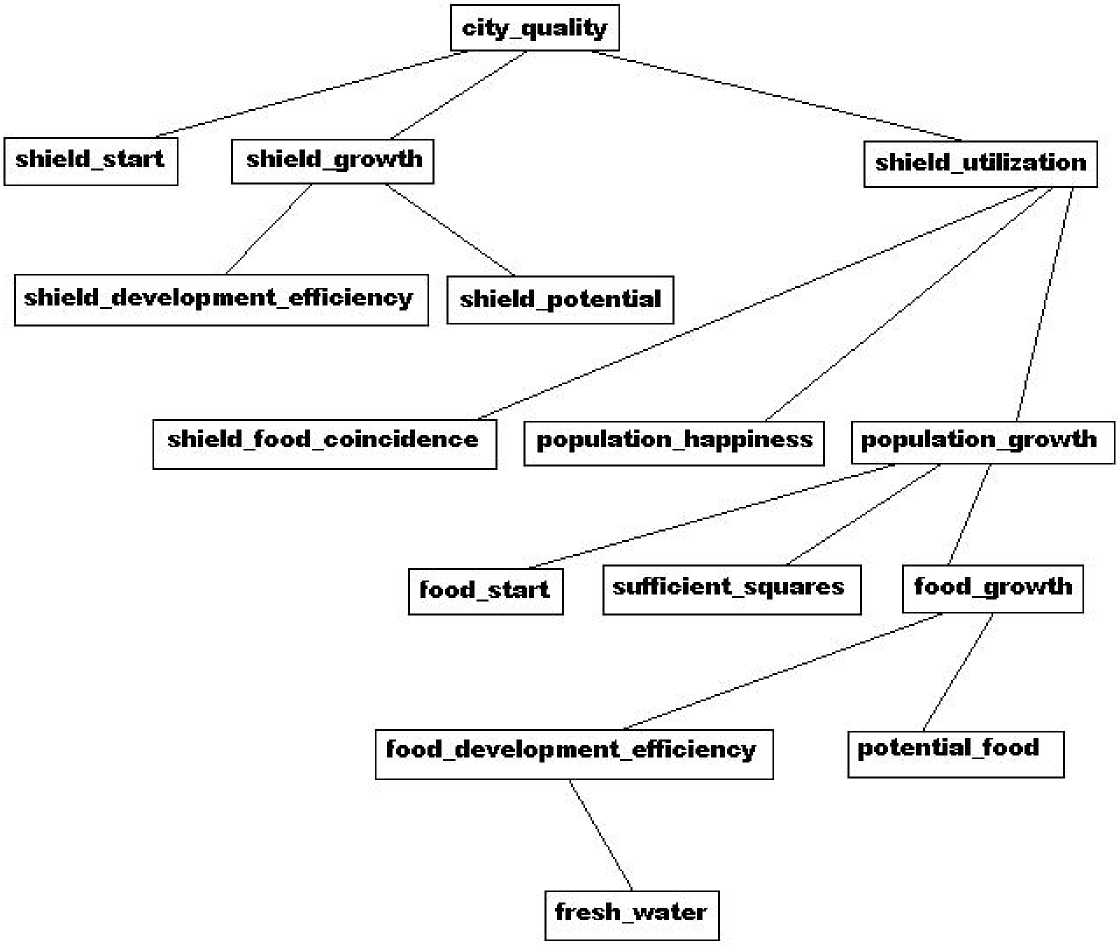


Figure 5: FreeCiv city production estimate classifier.

1.1.2 Compositional Classification

We formally define compositional classification as follows.

Let t be a discrete random variable representing the class label. Let $S = \{s : s$ is empirically determinable and $h[t] > h[t|s]\}$, where $h[x]$ denotes the entropy of x. S is a set of discrete random variables that have nonzero mutual information with the class label and are "empirically determinable." Each member s of S represents a related set of equivalence classes, where each value taken by s is a unique equivalence class. In the case of FreeCiv, things like the future population growth of the potential city and the amount of food provided by terrain squares around the city location constitute S. If, as above in the description of the general classification problem, we call F the set of features provided to the classification system *before* classification, we have $F \subseteq S$. A task instance is generated by jointly sampling the variables in $S \cup \{t\}$. In FreeCiv, the game engine handles this for us by randomly generating a game map and handling game dynamics that govern the relationships among the variables in S. Empirical determinability captures the notion of predictivity, indicating that each equivalence class represents some verifiable statement about the world. In the simplest case, empirical determinability means that the value taken by the variable in a given task instance is directly observable. In general, some experiment (a branching sequence of actions and observations) may need to be performed in order to observe the value of some $s \in S$. The simple case can be seen as a trivial experiment consisting of zero actions and a single observation. In FreeCiv, all of the values can be directly observed, though some (those members of S not in F) can be observed only after classification has occurred.

Each experiment has some nonnegative cost. We denote by $C_b(s)$ the cost of the experiment required to determine s before predicting the class label. The task is constrained by limited resources; only a fixed cost R_b may be incurred before the decision about the class label must be produced[1]. For this reason, the values of only a proper subset of S will in general be known when the prediction must be produced. Let $K \subseteq S$ with $\sum_{k \in K} C_b(k) \leq R_b$ be the information available at the time that classification must be performed. In the FreeCiv task, the resource constraint is time. In order to be useful, the prediction of city resource production must be made before the city is actually constructed and its resource production rate can be observed. Thus, we cannot directly observe the proper values of non-leaf nodes at inference time, but can obtain the true values later in order to learn.

Learning is required in part because the distributions $\mathcal{P}(s|K), s \in S \cup \{T\}, K \subseteq S$ are not assumed to be given, but must be inferred from experience. In this way, we are able to relax the requirements on the knowledge engineer constructing the agent's knowledge; if knowledge about the distributions is available a priori, it is possible to initialize the classification knowledge accordingly and decrease the demands on learning. But, when this knowledge is not available, not complete, or not correct, we require the system to learn the correct values.

After the predictive class label is produced and some time passes, the correct class label is determined and some additional quantity of resources R_a is allotted to the learner. These resources are then used to determine the values of other variables empirically before the next task instance is presented. The costs of performing experiments before predicting the class label may not be the same as the costs of performing experiments afterwards. For this reason, we denote by $C_a(s)$ the cost of performing experiment s after class label prediction. For some domains we may have $C_a = C_b$, but this need not be true in general. In the subclass of compositional classification problems addressed in this dissertation, there is a proper subset of S

[1] Actually, these costs and resources are better represented as vectors, as there may be multiple dimensions of cost, where each dimension has its own constraint.

that is always available before classification and the remainder of S is never available until after classification. This is a special case of the general domain, where $R_b = 0, R_a = \sum_{s \in S \cup \{t\}} C_a(s)$ and there is some (proper) subset of S s.t. $C_b(x) = 0$ for all x in the subset. This characteristic is important because it makes the value of information problem trivial. In this dissertation, we focus exclusively on problems with this characteristic in order to avoid the need to incorporate strategies for determining information value at this time. Generalization of the technique remains open as a potential direction for future work. Because this subclass of compositional classification problems is quite common, where the constraining costs are temporal and either 0 or ∞ before classification[2], this subset of problems is far from empty and is interesting in its own right. However, it is worth identifying the more general class of problems with arbitrary cost values on experiments because it is likely that techniques proposed for the problem subclass considered here can be extended to cover the general case by drawing from work on information value theory [49] [52] and budgeted learning [68]. Information cost is also taken into account in active learning [117], though the setting in active learning is different from that considered here (see Section 8).

Success at this learning task can be measured in terms of the final classification accuracy achieved, the rate at which accuracy improves as examples are presented, and the resources saved. Alternatively, resources saved during one instance could be made available for use during subsequent instances. In this case, the resources remaining at the end of the sequence contribute to the success measure. For the subclass of problems we consider here, we consider resource conservation to be a less important metric.

[2] Of course, there is also a class of problems where the constraints are temporal and take arbitrary real values. For instance, this occurs when the experiments that must be performed to determine values before classification are non-trivial, but there is some fixed time horizon in which a decision must be made. Such problems once again require judgements about value of information, and thus are not examined here.

As a secondary consequence of the choice of compositional classification problems as the problem setting in which to test the use of EVPs for adaptation of domain knowledge, this research can be seen as having some impact on two simultaneous problems within compositional classification. First is the problem of enhancing the efficiency in terms of time and/or number of required examples of existing numerical techniques used for compositional classification problems. Second is the problem of making efficient knowledge-based techniques for compositional classification more flexible by adding the capacity for automatic diagnosis and repair.

Another secondary contribution of this research is in presenting this particular framing of the classification problem. This work takes a strongly agent-based view of the classification learner – we view the learner as operating within a dynamic environment that supports a rich variety of percepts and actions. As this environment evolves over time, the agent can interact with it to gather information relevant to past decisions. This stands in contrast to a more classical view of the classification problem, where interactions within the environment are narrow and fixed (often restricted to only receiving the correct class label for a given example). Thus, learning is both an introspective and extrospective process, where "extrospective" indicates looking outside oneself, in contrast to introspective processes. Under this paradigm, the agent examines and diagnoses its own knowledge in light of evidence gathered from the environment – the environment to which that knowledge pertains. The following chapter describes both the representation used to capture this classification knowledge, and the diagnostic procedures used to support its revision.

1.2 Results and Claims

- Empirical Verification Procedures allow a reflective process to successfully identify and correct faulty knowledge within a hierarchical classification structure. (Chapter 4)

- Empirical Verification Procedures associated with nodes in a classification hierarchy impose a restriction bias on the hypotheses representable by that knowledge structure, and therefore increase the generalization power of a learner that makes use of the representation. (Chapter 5).

- Empirical Verification Procedures can be automatically adjusted when the concepts they define are not successful in their functional roles. (Chapter 6)

There are also a number of secondary results that arise due to the choice of compositional classification as a domain in which to test the use of EVPs for domain knowledge adaptation:

- Formal and empirical demonstrations of the benefit of using structured knowledge representations for classification. (Chapter 4)

- Empirical evidence that the performance of learners using hypothesis spaces limited by hierarchical classification structures degrades gracefully as errors in the structure of the hierarchy are introduced. (Section 4.3)

- A formal demonstration of the conditions under which a "causal backtracing" diagnostic procedure is optimal. (Section 7.1)

- A description of a design space, comprising a set of parameters and constraints, within which choices must be made when applying hierarchical classification to a compositional classification problem. (Chapter 9)

1.3 Dissertation Outline

The remainder of this dissertation is structured as follows:

2. **Applying Reflection to Compositional Classification Knowledge**: Provides a formal description of the key data structures and algorithms in this work.

3. **Experimental Design**: Describes the set of experiments we have performed with Augur.

4. **Experimental Results**: Presents technical details and analyzes the outcomes of experiments described in Chapter 3.

5. **Learning with Unspecified Concept Semantics**: Describes empirical results with hierarchical classification learners that have fixed semantics at only *some* internal nodes, rather than at all nodes as in most of the work described here.

6. **Automatic Concept Refinement**: Describes the mechanisms implemented to automatically refine concepts within a hierarchical classifier, and presents experimental results.

7. **Analysis of Compositional Classification**: Details some results relevant to hierarchical classification.

8. **Related Research**: Connects the research with relevant topics in Artificial Intelligence and Machine Learning.

9. **Conclusion**: States the claims and contributions of the work, both technical and conceptual.

Together, these chapters form the basis for a theory of reflective adaptation of domain knowledge and an instantiation of that theory within the problem setting of compositional classification.

CHAPTER II

APPLYING REFLECTION TO COMPOSITIONAL CLASSIFICATION KNOWLEDGE

In this chapter, the representational structures used to address the compositional classification problem and the reasoning processes that operate over those structures are described formally. We will begin with the central structure in this work, the Empirical Verification Procedure, which we believe has applications for metareasoning-based adaptation of domain knowledge beyond the scope of Abstraction Networks, and even beyond compositional classification problems.

2.1 Empirical Verification Procedures

Definition 1 *An* Empirical Verification Procedure *is a tuple $\langle E, O, C_b, C_a \rangle$ where O is a set of output symbols (output space) and E is a possibly branching sequence of actions in the environment and observations from the environment concluding with the selection of an $o \in O$. C_b and C_a are the costs of procedure E before and after classification, respectively.*

We can now be more specific about what makes a set of equivalence classes empirically determinable, a term used more informally in the description of compositional classification in the prior chapter. Any output space O of an Empirical Verification Procedure is an empirically determinable set of equivalence classes. So, viewed from the other direction, a set of equivalence classes is empirically determinable if an Empirical Verification Procedure can be defined with an output space equal to that set of classes. Note that this definition is in terms of the actions and observations available to the agent that learns and reasons with the knowledge, making a commitment about the way that interaction with the environment is expected to justify and give meaning to knowledge in this system.

2.1.1 Taxonomy of EVP Types and Related Adaptations

The formal definition of EVPs given above remains silent about the types of actions and observations that constitute E. If one does not wish to operate upon E, but simply execute it to verify the application of associated knowledge, this definition is sufficient. As long as there is some way to execute the EVP and retrieve the result, the kinds of operations performed are not terribly important from a learning perspective. However, as noted in Chapter 1, one of the benefits of explicitly representing conceptual semantics is that those semantics can then themselves be operated upon directly by the agent, and automatically adjusted. However, if we wish to encode procedures for such operations, it becomes important to know more about the kinds of operations that may be performed by EVPs, and how those procedures might be adjusted. This section addresses these questions. Following is a taxonomy of operation types that may be performed within an EVP. While this taxonomy is not necessarily exhaustive, it is sufficient to cover all of the EVPs used in the work described in this dissertation, and appears likely to be sufficient for a wide range of applications.

- **Act and Continue**: Take some action in the environment and continue to the next operation in the EVP.

- **Observe, Branch and Continue**: Make some observation from the environment and conditionally branch based upon the percept's value, continuing to the next operation in the EVP along the selected branch.

- **Emit Category**: Return the value that would have properly predicted the environmental situation measured by this EVP, and terminate.

- **Fail**: Abort and terminate, producing no value.

All branches within EVPs based on these primitives will terminate with either 'Emit Category' or 'Fail' operations. A limited number of potentially useful ways to modify EVPs composed of such building blocks suggest themselves:

- Alter EVP composition, e.g. insert an 'Act' or an 'Observe/Branch' along with new children.
- Adjust the conditions tested within a branch, e.g. change a perceptual threshold used to choose one branch over another.
- Alter the number of outputs of a branch, e.g. add a new branch choice, adjusting branching conditions such that the new branch may sometimes be selected.

A particular sub-case of the third modification listed above, specifically increasing the fanout of a branch that leads directly to 'Emit Category' children, has been implemented. Empirical results are discussed in Chapter 6. This modification was of particular interest in this work because many of the EVPs used when experimenting in the real domains, and all of the EVPs used in the synthetic domain, are of a particular type which we call *quantizing EVPs*. These EVPs consist of a single 'Observe, Branch and Continue' operation which leads directly to 'Emit Category' children. Adding children to the 'Observe, Branch and Continue' operation (and making a corresponding adjustment to the branching conditions used by the operation) has the effect of altering the classification made by the node such that more information about child values flows through the node – conversely, less information is lost by the classification being made at the node. The consequences of adjusting the information loss within nodes in an Abstraction Network are also discussed more formally in Section 7.3.

2.2 Abstraction Networks for Compositional Classification

We will now move to the definition of the hierarchical classification structures used for compositional classification specifically. EVPs, described in the previous section, will be used to semantically ground concepts within the Abstraction Networks, and will become crucial in self-diagnosis when classification failures are detected. Informally, we begin by establishing a node for each $s \in S \cup \{t\}$. These nodes are connected according to the given dependency structure, which we know will result in a hierarchy based on the given assumptions. This structuring follows the pattern of structured matching [12] [37]. A structure used for experimentation in the previously discussed FreeCiv problem is depicted in Figure 5. Each node will handle the subproblem of learning to predict the value of the variable with which it is associated given the values of its children, which are the variables upon which the variable to be predicted has direct (forward) dependency. Organizing the structure of the knowledge to be learned in this fashion has the benefit of making full use of the dependency structure knowledge to limit the hypothesis space while being certain not to eliminate any hypothesis that could be correct, and also yields the proven efficiency benefits of hierarchical classification [37].

A more formal definition follows.

Definition 2 *Here, we will define a* supervised classification learner *as a tuple* $\langle I, O, F, U \rangle$*, where* I *is a set of input strings (input space),* O *is a set of output symbols (output space),* F *is a function from* I *to* O*, and* U *is a function from* $(i, o) : i \in I, o \in O$ *to the set of supervised classification learners that share the same input space* I *and output space* O*.* U *is an update function that has the effect of changing* F *based upon a training example.*

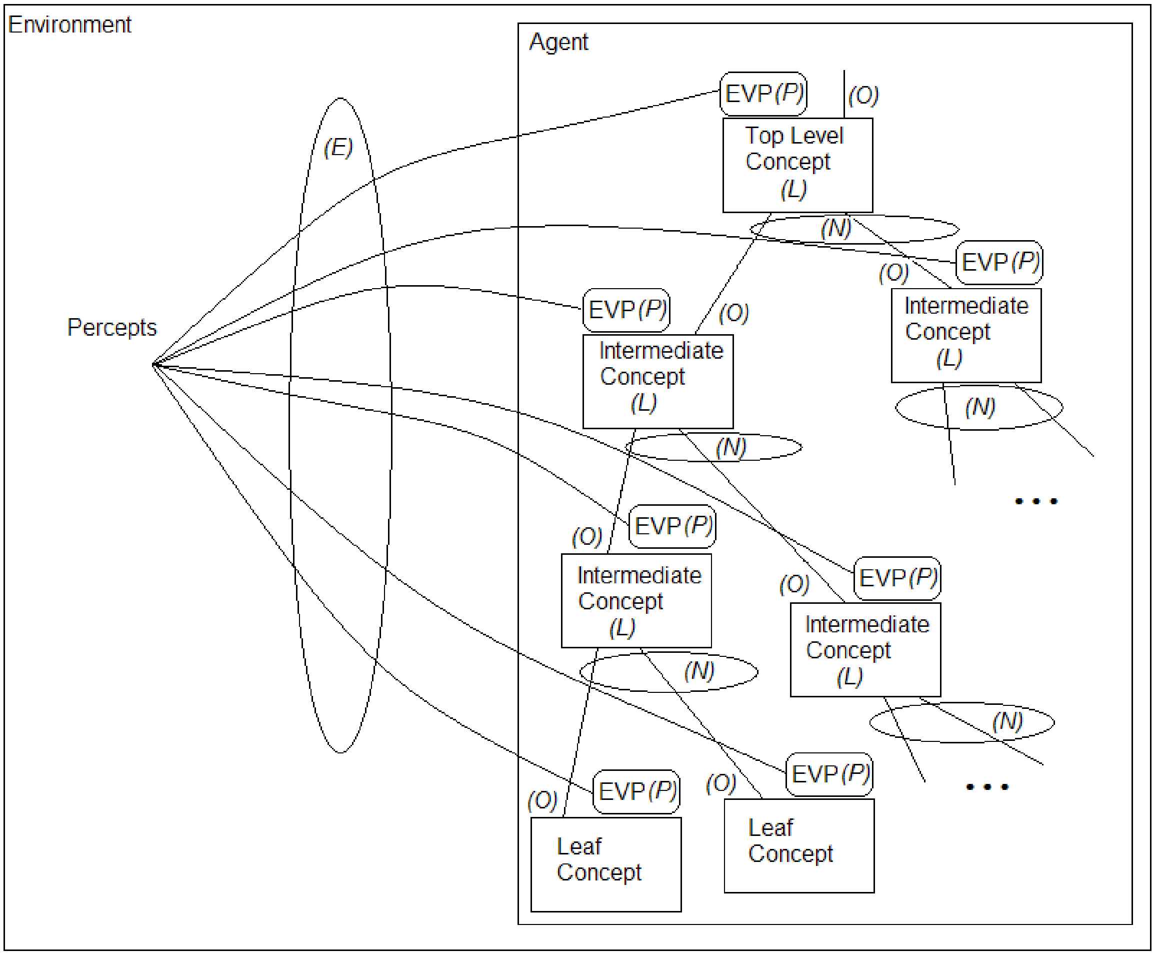


Figure 6: General Abstraction Network architecture with annotations from Definitions 1 and 3.

Definition 3 *An* Abstraction Network *is recursively defined as follows. A tuple* $\langle \emptyset, O, L, P, last\_input, last\_value \rangle$ *is an Abstraction Network, where* O *is a set of output symbols,* L *is a supervised classification learner, and* P *is an Empirical Verification Procedure.* $last\_input$ *and* $last\_value$ *are used to cache input and return values at AN nodes in order to support the learning procedure (detailed below). A tuple* $\langle N, O, L, P, last\_input, last\_value \rangle$ *is an Abstraction Network, where* N *is a set of Abstraction Networks. Let* I *be the set of strings formable by imposing a fixed order on the members of* N *and choosing exactly one output symbol from each* $n \in N$ *according to this order. The supervised classification learner* L *has input space* I *and output space* O*, and the Empirical Verification Procedure* P *has output space* O*.*

Notice that this definition requires Abstraction Networks to be trees, rather than some more general structure such as DAGs. Some of the algorithms used in this work do not admit DAGs (e.g. the automatic concept refinement procedures of Chapter 6). It is possible that this work could be generalized to work over more general data structures, but this is left as future research. Note also that each AN node contains its own supervised classification learner. This means that both learned concept identification knowledge and *the learning algorithm* can in principle be selected on a per-node basis. That is, there is nothing in the definition of ANs that requires homogeneity in the learning algorithms used within nodes in an AN. However, we have not performed experiments with mixed learner types. Figure 6 shows the general AN architecture with annotations from Definitions 1 and 3.

When N is empty, L is trivial and has no use as the input space is empty. In these cases (the leaves of the AN), the only way to make a value determination is to invoke P. Because the subproblem considered in this dissertation is restricted to cases where AN leaves are always determined empirically before classification, this is not an issue. That is, in the current work, whenever $N = \emptyset, P.C_b = 0$. If the technique is generalized, provisions will have to be made to deal with undetermined leaf values. Having described the AN representation, we next turn to reasoning (performing predictive classification) using an AN.

2.3 Reasoning

In a given task instance, the values of the leaf nodes are fixed by observation. As described above, in the problem settings considered here, obtaining the values of the leaf nodes has zero cost, and no other values are available before classification. Each node with fixed inputs then produces its prediction. This is repeated until the value of the class label is predicted by the root of the hierarchy. This procedure will produce the most likely class label based on the current state of knowledge.

The reasoning procedure over an arbitrary AN a is more formally described in Table 1. All fields referenced using the "dot" notation use the names from the definitions of the previous section.

2.4 Self-Diagnosis and Repair

At some time after classification, the true value of the class label is obtained by the monitoring process. If the value produced by object-level reasoning was correct, no further action is taken. If the value is found to be incorrect, a self-diagnosis and repair procedure is followed. The specifics of this procedure are dependent upon the characteristics of the learner types that are used within nodes and the classification problem setting. For most of the empirical results detailed in this thesis, the following procedure is used, beginning with the root of the hierarchy as the "current node" when external feedback indicates that the top level value produced was incorrect:

1. The true value of each child of the current node is obtained by executing the associated EVPs.

2. If the predictions of all children were correct, modify local knowledge at the current node.

Table 1: Reasoning procedure used to produce a predictive classification from an Abstraction Network a.

```
/* Values from Definition 3:
 * a.N              - a set of ANs.  The children of ‘a’.
 * a.P              - the EVP for ‘a’.
 * a.last_input     - the last input sequence provided to ‘a’.
 * a.last_value     - the last value produced by ‘a’.
 * a.L              - the learner associated with ‘a’.
 *
 * Values from Definition 2:
 * L.F              - the learner’s inference function.
 *
 * Subfunctions used:
 *   push_back(String $i$, Value $V$):
 *                  Appends the value provided as the second argument
 *                  to the string provided as the first.
 */

begin AN-reasoning(Abstraction Network $a$)
      String $i$

      /* If we are at a leaf, return the result of executing the local
       * EVP, which for the domains considered here, is always possible
       * at leaves.  These values are the ‘‘inputs" to the AN inference
       * process.  */
      if $a.N = \emptyset$, return $a.P$

      /* Otherwise, build the input vector for the local learner
       * and return the result of applying it.  */
      forall $n \in a.N$:
               push_back(i,AN-reasoning($n$))
      $a.last\_input \leftarrow i$
      $a.last\_value \leftarrow a.L.F(i)$
      return $a.last\_value$
end
```

Table 2: Self-diagnosis and self-repair procedure used to correct knowledge stored in an Abstraction Network *a*.

```
/* Values from Definition 3:
 * a.P              - the EVP for ‘a’.
 * a.last_value     - the last value produced by ‘a’.
 * a.N              - a set of ANs.  The children of ‘a’.
 * a.L              - the learner associated with ‘a’.
 * a.last_input     - the last input sequence provided to ‘a’.
 *
 * Values from Definition 2:
 * L.U              - the learner’s update (learning) function.
*/

begin AN-learning(Abstraction Network $a$)
      Bool $flag \leftarrow true$
      if $a.P() = a.last\_value$, return $true$
      forall $n \in a.N$
               if AN-learning($n$) $= false$, $flag \leftarrow false$
      if !$flag$, return $false$
      $a.L \leftarrow a.L.U((a.last\_input, a.P()))$
      return $false$.
end
```

3. Otherwise, recursively repeat this procedure for each child node that was found to have produced an incorrect prediction.

The procedure for self-repair and self-diagnosis, for an AN *a*, is more formally described in Table 2, and illustrated in Figure 7 (note that *last_value* and *last_input*, used in Table 2, are explained in Section 2.3 above).

Notice that this procedure has a base case when the leaves are reached, as their true values were obtained before classification, and thus cannot be found to be incorrect during learning. Also note that some optimizations to the procedure of Table 2 are certainly possible; for instance, if the procedure finds that a given node’s children have produced errors, the procedure *could*, after attempting to repair lower-level errors,

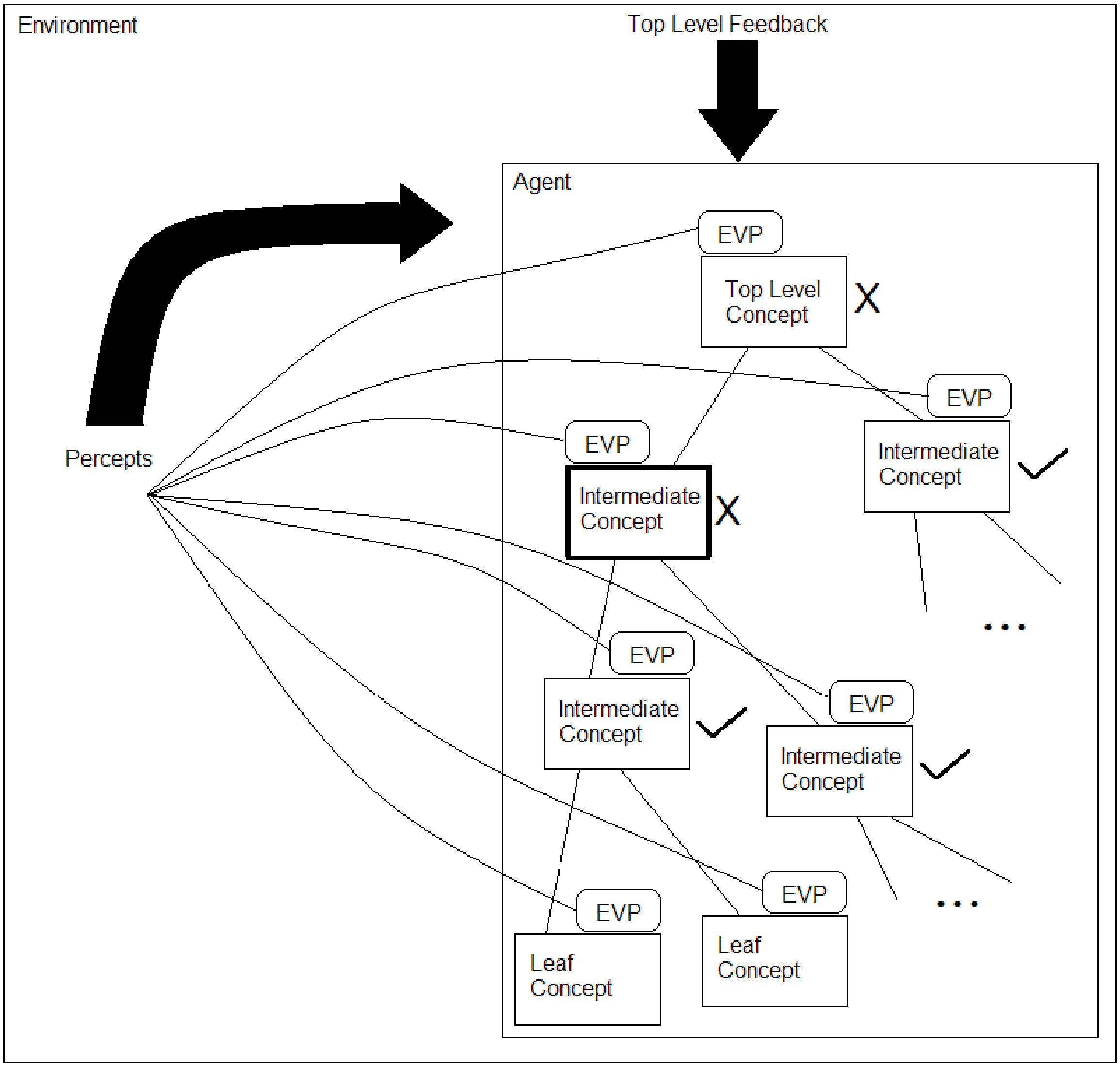


Figure 7: An example outcome of the diagnostic procedure of Table 2. First, top level feedback indicates a problem with the overall classification. Then, EVPs within the hierarchy are progressively executed, resulting in the examination of various percepts. Nodes marked with an "X" were found to have produced incorrect values after EVP execution, while those with checkmarks were found to be correct. No EVPs beyond those associated with nodes for which results are shown would be executed in this case, as diagnosis has located a frontier of correct nodes. Local learning in this case would occur at the node with a bold border, as it is the only incorrect node with correct children.

then check to see whether the higher level node would have produced the correct value given correct inputs from the children. If not, the procedure could then learn at that higher level node as well. However, the experiments run in this dissertation do not make use of such conceivable optimizations.

Depending upon the relative weighting of resource preservation and learning speed in the evaluation metric, this procedure may be suboptimal, because it implements the policy of only obtaining information when it is certain that the information will lead to learning (with the exception of the class label, which is always determined after prediction). It is likely to be a good policy when resource preservation is weighted highly against learning speed, and a poor policy when the reverse is true. Some tuning of the balance between obtaining the correct values of more nodes and preserving resources could be a useful generalization of the technique. However, note that this choice of policy will not prevent convergence. If we are to consider some piece of knowledge stored within the hierarchy as incorrect, there must be at least one situation occurring with nonzero probability where that knowledge will lead to an incorrect overall result (we take this as the definition of incorrect knowledge). When this situation arises, the incorrect knowledge will be identified by credit assignment, and the knowledge will be modified.

2.4.1 Discussion

One point to notice here is that the specific procedure for the modification of local knowledge is not specified. Any supervised classification learner that satisfies the definition given in the Section 2.2 is acceptable. A closely related point is that the representation of the knowledge, and thus the procedure for knowledge application within each node, is similarly unspecified. This is quite intentional: *any* knowledge representation/inference/learning technique can be used within each node. Heterogenous sets of techniques could in principle be used within a single hierarchy. The specific technique or set of techniques that will perform best on a given problem depends on the specifics of the subproblems – choosing learning techniques that exploit known characteristics of each subproblem will, of course, lead to the best overall results. For instance, for some kinds of problems it may be that Bayesian learning of probabilities is the most effective technique at all nodes. In this case, the overall learner is somewhat similar to a particular type of Bayes net, augmented with a learning procedure that is sensitive to knowledge acquisition costs. See the discussion on related research in Section 8 for a more thorough comparison with Bayes nets, and Appendix B for an empirical comparison. In other cases, it may make sense to use Artificial Neural Networks (ANNs) [94] within some or all nodes, in order to introduce a different kind of inductive bias (based on smooth interpolation) for some subproblems. Generally, the point is that because the characteristics of the dependencies between members of $S \cup \{T\}$ are not fixed over the entire domain of interest, it does not make sense to fix a learning method for the subproblems in the absence of knowledge, nor is it necessary to do so in order to specify a solution exploiting domain characteristics that *are* given. Of course, when instantiating this technique for a specific domain, these choices must be made.

It is important to note here that, based upon the choice of learner type(s) to be used within an AN, other choices such as the diagnostic procedure to be followed may be constrained. For example, some learner types such as ANNs depend upon training examples being drawn from a stable distribution. The diagnostic procedure discussed in this section cannot make such a guarantee. However, we have identified at least one simple diagnostic procedure that can make this sort of guarantee: execute all EVPs within an AN for each diagnostic episode, performing this operation even when the top-level classification was found to be correct. This linked pair of choices, learner type and diagnostic procedure, illustrates a tradeoff that must be considered when a designer is instantiating an AN for a specific problem. Is it more important to use a learner type with a particular bias? Or to use a diagnostic procedure that parsimoniously executes EVPs? The answer will depend upon the relative costs of example acquisition and EVP execution within the domain addressed. We return to the discussion of these issues in Section 7.1, which demonstrates the parsimony of this section's diagnostic technique under certain assumptions.

To again return to the comments at the end of the last chapter, where differences between the agent-oriented view of classification learning adopted by this work and the more classical view were highlighted, notice that in learning techniques based upon the classical view, much of the knowledge engineering effort is implicitly represented from the learning agent's point of view, hidden, for example, in the construction of the feature set. In contrast, this work calls for the explicit representation of the connection between equivalence classes used by the classifier and raw perception. By including an explicit representation of these abstractions, we both enhance the inspectability of the knowledge structures used for classification by effectively annotating intermediate nodes with semantics[1], and also allow these abstractions themselves to be directly operated upon during learning. In this vein, in addition to the content learning procedure described above, where the knowledge contained at nodes within the network is modified, we have also done some work on automatically tuning the equivalence classes at nodes within the hierarchy. This translates to automatically adjusting O, the set of output values, at nodes within the hierarchy, adjusting the concepts represented by those nodes. Changing the set of output values at a node hinges on the ability to adjust the node's EVP, as noted above in Section 2.1, such that newly added output values will be learned to apply to some set of situations, or deleted output values will be learned to be obsolete. Finding the right level of

[1] This feature of the EVP-based metareasoning approach to classification learning is also likely to have other benefits, such as facilitating portions of an AN structure trained on one top-level problem to a different top-level problem. This potential benefit also stems from the fact that under EVP-based learning, each portion of the AN trained has known semantics.

information loss at nodes in the hierarchy is important because too much loss (i.e. too few output values) will provide insufficient information to the parent node and cause learning failure, while too little information loss decreases the generalization power of the network for learning (see analysis in Section 7.3). At the limit, if there is no information loss at any node in the network, the representation becomes equivalent to a flat representation. Because too much information loss results in learning failures, identifying these failures will help to identify locations at which information loss should be decreased by increasing the available output values. Given this capability, we can start with very high information loss at each node and allow loss to be decreased as breakdowns of the learning process are identified. Our experiments using such a procedure are described in Chapter 6.

CHAPTER III

EXPERIMENTAL DESIGN

In this chapter, we describe the set of experiments that we have performed with the Augur system primarily in order to test the efficacy of EVPs in allowing an agent to reflect upon and adapt its own domain-specific classification knowledge. Given that we are working within the setting of compositional classification, many of these experiments also provide results relevant to characteristics of compositional classification and particularly the use of hierarchical classification knowledge structures to learn within the problem setting. The usefulness of EVPs is supported by each of the experiments, which demonstrate the generality of the usefulness of EVP metaknowledge along several dimensions:

- **Types of learners** used within nodes: We have experimented with rote table-based learners, k-Nearest Neighbor learners and Artificial Neural Networks operating within AN nodes.
- **Problem domain**: We have experimented in the game FreeCiv, a Dow Jones Industrial Average prediction problem, and a sports prediction problem as well as a synthetic domain.
- **Quality of knowledge engineering**: We have systematically degraded the quality of knowledge engineering in two ways, by removing individual nodes from an AN hierarchy and by removing entire subtrees.

These experiments, taken together, demonstrate the effectiveness of EVP-based reflection over and adaptation of metaknowledge under a variety of conditions within the general domain of compositional classification. As such, they support the first major hypothesis put forth in Chapter 1, that Empirical Verification Procedures do allow a reflective process to successfully identify and correct faulty knowledge within a hierarchical classification structure. Further, they demonstrate some characteristics of Abstraction Networks, or more generally, hierarchical classification knowledge structures:

- Structured knowledge representations provide a substantial restriction bias on a learner's hypothesis space, increasing generalization power and thus requiring fewer training examples to reach minimum error. This benefit is independent of the learner type used within nodes in the classification hierarchy.
- The performance of learners using hypothesis spaces limited by hierarchical classification structures degrades gracefully as errors in the structure of the hierarchy are introduced.

We also wish to determine whether, beyond the hierarchical knowledge representation integral to ANs, there is also generalization power imparted by the use of EVPs. EVPs essentially fix, or *pin* the semantics of nodes within an AN structure by defining the appropriate values that should be produced in any situation. Intuitively, this kind of semantic pinning should reduce the number of hypotheses that are expressible by a knowledge representation, and as such, represent a restriction bias that increases generalization power. We test this hypotheses in the experiments of Chapter 5, where we compare the performance of ANs that have had EVPs removed from some of their nodes with ANs that have EVPs at all nodes.

Another very key set of experiments, that we describe in Chapter 6, speaks directly to another particular sense in which EVP metaknowledge is useful. In the experiments of this section, the definition of some concepts in an AN learner do not, by design, provide adequate information to their parents. However, we have implemented an automatic mechanism by which this deficiency may be detected through reflection, and automatically corrected. These experiments demonstrate directly the power that is gained by imbuing a system with explicit, first class knowledge of both the meaning (via EVPs) and functional role (via AN structure) of its knowledge. Regardless of the relative merits of other classification systems (which in many cases could likely be integrated with both ANs and EVPs, or at least the latter), the capability to automatically operate on concept semantics *cannot* be achieved without explicit, accessible representation of those semantics – and this is the core idea of the EVP. This set of experiments provides evidence to support the second major hypothesis proposed in Chapter 1, that Empirical Verification Procedures can be automatically adjusted when the concepts they define are not successful in their functional roles.

As a secondary set of results, we also provide an empirical comparison of the performance of Bayesian Networks (BNs) vs. that of Abstraction Networks under a variety of conditions within the synthetic domain. These experiments are presented in Appendix B. While these experiments have no bearing upon the claims made in this dissertation, as EVP-based reflective learning could be used equally well within the context of BNs as ANs, the results are somewhat interesting in their own right and so are included for the sake of completeness. These experiments demonstrate some advantage in terms of AN learning rate and computation time as problem size increases. It is likely that these advantages are due to the fact that ANs pass less information from node to node during inference (ANs commit, in effect, to the most likely value at each node, while BNs pass a distribution over all values). It is clear why this difference between ANs and classical BNs would lead to a difference in computational effort during inference, and it seems likely that observed differences in learning rate are similarly attributable. Thus, one could likely match the performance of ANs in these experiments by using BNs that commit fully to the most likely value at each node during inference (setting the probability of the most likely value to 1 and all other values to 0). However, once again this experiment is quite peripheral to the thesis here, as EVPs could equally well be used in BN learning.

3.1 Rote Learners

As noted above, we have integrated three types of learners with the Augur system, and we have performed some experiments with each of them. While ANN and kNN learners are well known (we describe relevant parameters in detail in Chapter 4), in this section we define the table-based rote learners with which we also experiment. These table-based rote learners are an instance of the supervised classification learners of Definition 2. Essentially, these rote learners simply maintain a table mapping from input combinations directly to output values. There is a bit more complexity in these learners that stems from the desire to imbue them with some modicum of robustness in the face of noise.

Definition 4 *A* rote learner *is a tuple $\langle I, T, O, F, U\rangle$, solving a classification problem that requires mapping a finite input space I onto a finite set of contiguous integers O. T is a finite set of contiguous integers that is symmetric about zero, and we call $(|T|-1)/2$ the "learning threshold" of the rote learner. F is a function from I to O, implemented as a composition of two functions, F_1 and F_2. F_1 maps from I to $O \times T$ and F_2 maps from $O \times T$ to O. F_2 is defined such that $\forall t \in T, o \in O, (o,t) \to o$.*

Given an input example (i, o'), U returns a new rote learner $\langle I, T, O, F', U'\rangle$ where F' is a composition $F_2 \circ F'_1$. $\forall x \neq i, F'_1(x) = F_1(x)$. Let $F_1(i) = (o,t)$. Then, if $t+(o'-o) \in T$, $F'_1(i) = (o, t+(o'-o))$. Otherwise, if $t+(o'-o) < 0$, $F'_1(i) = (o-1, 0)$ or if $t+(o'-o) > 0$, $F'_1(i) = (o+1, 0)$. U' is an update to U to embed knowledge of the new function F' such that the next update can proceed by the same logic.

Informally, the rote learner requires indication of a significant error (an (i, o') where o' is quite different from $F(i)$) or demands some consistency in feedback before making a change to the classification of a given input. The purpose of F_1 is to record the amount of error seen so far with respect to a particular input sequence. The purpose of F_2, then, is simply to strip this error information away and return the desired output value. At each update, the update function U checks whether the error threshold has been exceeded by looking at the information recorded for the appropriate input sequence by function F_1, updating the output value only if the threshold has been exceeded in either the positive or negative direction. Otherwise, the error measure is updated but the output value for the input sequence is not. The intent of this scheme is to avoid making changes due to noise in input examples. In the synthetic domain, noise is not an issue, but becomes significant in the additional domains reported on below. Table update learners as described here are only sensible if there is some natural ordering of class labels; otherwise, taking the difference of class labels is not meaningful. However, this limitation as well as the overall simplicity of rote learners is not a necessary aspect of ANs in general. We have used these very simple learners in order to demonstrate the power of the framework itself. In this work, all rote learners in each experimental setting used a threshold of 5, except for the comparison with Bayesian networks discussed in Chapter B.

Similar rote learners are discussed by Kohavi in [62]. These learners share the same basic principle as those used in this work – memorize input examples. However, there are some key differences. First, Kohavi's rote learners record *all* input examples, returning the most common output label seen for training examples with input features that match those of a query. In contrast, the rote learners described here maintain no such history of examples, but rather record only the classification label consistent with most recently seen examples. This difference could impact the way that these two types of rote learners react in dynamic environments, though either will eventually adapt to a changed environment. Second, Kohavi's learners also maintain a global "most common output class" that is returned when a query does not match the input features of any observed training example. On the other hand, the learners described here would return a random initialization value when confronted with an input example for which no relevant raining examples have been seen. Though relatively minor, this added feature of Kohavi's learners provides a modicum of generalization power that the rote learners of this work lack. Finally, and most importantly, Kohavi's work described in [62] illustrates the generalization power imparted on rote learners by a principled process of feature selection. His learning system generalizes over training examples by selectively discarding features that are found statistically irrelevant to the output class. The rote learners used in this work are imbued with no such automatic feature selection procedure, and have no generalization power of their own. All of the generalization power of ANs using rote learners comes from the hierarchical knowledge structure (demonstrated in Chapter 4) and from the semantic pinning of nodes within the hierarchy via EVP (demonstrated in Chapter 5).

In the next three chapters (4, 5 & 6) and Appendix B we describe technical details of our experiments with the Augur system and analyze the results in detail.

CHAPTER IV

EXPERIMENTAL RESULTS

In this chapter, we detail empirical results that have been obtained to test several aspects of EVP-based self-diagnosis and learning. These experiments also demonstrate some characteristics of Abstraction Networks. All of these experiments make use of the Augur system's AN implementation. Results are presented in a synthetic problem, as well as in three non-synthetic instances of the compositional classification problem. Results include tests with table-based rote learners, Artificial Neural Networks (ANNs) [94] and k-Nearest Neighbor learners (kNNs) [28] working within AN nodes. Beyond experiments that test the central hypothesis of this thesis, that EVPs provide adequate metaknowledge for an agent to self-diagnose and repair faults in its classification knowledge (Sections 4.1 & 4.2), we also describe experiments dealing with the effects of faulty structural knowledge engineering on AN performance (Section 4.3).

4.1 Synthetic Domain

In order to verify that EVP-based self-diagnosis does allow for correction of faulty knowledge engineered content in an AN and to demonstrate some degree of generality of ANs with respect to the learner types used within nodes, we have performed a set of experiments in a synthetic domain. The environment in this domain consists of a fixed Abstraction Network, over which no learning will occur, that represents the correct, target content (and structure) for the problem. Given this fixed AN, we then create a separate *learner* AN that will be initialized with incorrect knowledge content and expected to learn to functionally match[1] the content of the target AN. This is implemented by initializing the knowledge content of both the fixed and learner AN nodes separately with pseudo-random values. The randomly-generated content of the fixed AN forms the target knowledge for the learner AN. Because the work described here is concerned only with repairing content and not structure, we do build the learner AN with a correct structure that matches that of the fixed AN. Training proceeds by repeating the following steps:

1. Generating a pseudo-random sequence of floating point numbers to serve as the observations for the input nodes of the ANs.

2. Performing inference with the fixed AN, saving the values produced by all intermediate nodes as well as the root node.

3. Performing inference with the learner AN.

4. Performing EVP-based self-diagnosis and learning over the learner AN according to either the procedure described in Section 2.4 for table-based rote learners and kNN learners, or by executing all EVPs within the learner AN in the case of ANN learners within nodes.

There is another small adjustment to this procedure in the case of ANN learners within nodes, where we wish to use a batch-style training set/test set approach rather than sampling training examples continuously, as this is more traditional for ANN learning. This is described in more detail below in Subsection 4.1.2. In all cases in the synthetic domain, EVPs within the inputs of both ANs are set up to quantize the floating point observations. EVPs are not needed at non-leaf nodes in the fixed AN, since no learning will occur. EVPs at non-leaf nodes in the learning AN are set up to

[1] By "functional matching", we mean that we will only measure error based on whether the learner AN is able to produce output values that match those of the fixed AN – we will not actually inspect the contents of individual nodes.

examine the saved output value from the corresponding node in the fixed AN. In the first set of experiments we used simple table-based rote learners within each node.

4.1.1 Rote Learners

In this section, we describe results in the synthetic domain using rote learners (defined in Section 3.1) within the nodes of an AN learner. Each rote learner used a threshold value of 5. In the experiments in the synthetic domain, all of the structured ANs take the form of binary trees (each non-leaf node has a fan in of two). Every node, including the leaves and the root, chooses from among 3 possible output values in this set of experiments. Thus, each table update learner used in structured learners in the synthetic domain has $3^2 = 9$ entries, while the flat learner has 3^{inputs} entries. This set of experiments using rote learners includes three problem sizes. The largest has 16 inputs, with the binary structure yielding 8, 4 and 2 nodes at each subsequent layer. The other two problems use 8 and 4 inputs, respectively.

In addition to verifying that EVP-based self-diagnosis allows for correction of faulty AN content, we wished to empirically illustrate the benefit of using a structured knowledge representation matching domain structure vs. using a “flat”, unstructured representation. We also present formal results pertinent to this question in Section 7.3. Thus, in addition to the learner AN described above, we also trained a flat learner in each problem setting for which we report results in the synthetic domain. These flat learners are implemented as ANs where the input layer is connected directly to the output node. Thus, in the flat learners used in these experiments, there is a single rote learner that must learn the full mapping from inputs to output values without the generalization enabled by a structured representation. Results in each tested configuration are reported for both a structured AN learner and a flat learner.

We train and evaluate these learners in an on-line, incremental fashion, evaluating the learners’ performance improvement during training by segmenting the sequence of examples into multi-example blocks and comparing overall error rate between blocks. An error is counted whenever the learner's output on a given example does not match the output produced by the fixed AN. In this way, we are able to compare error rate around the beginning of a training sequence with the error rate around the end of that sequence. As noted in the previous section, this set of experiments uses the non-exhaustive diagnostic procedure described in Section 2.4. This means that in general, not all EVPs within the learner AN will be executed for a given example. Under this procedure, diagnosis immediately returns without performing learning if no error is detected at the AN root. In this domain, as in other domains, we first expect the learner AN to produce a prediction, and then subsequently expect more information to become available to allow the diagnostic procedure to be run (i.e. for EVPs to be executable).

The results of these experiments for the three synthetic domain sizes are depicted in Figures 8-10 in terms of per-block error rate. The results shown are an average of 100 independent runs in each setting, with separate random table initialization at the beginning of each run. Randomly initializing the tables in the generator ANs means randomly selecting an output value for each input combination. This process can lead to complex functions being produced by each generator AN node. Each run in the large problem setting consists of 10,000 generated examples, which we segment into 100 blocks of 100 examples for the purposes of visualization. In the medium-sized problem setting, 100 blocks of 50 examples were used in each run. Finally, in the small problem setting, each run consisted of 100 blocks of 10 examples each. These results demonstrate the efficacy of EVP-based self-diagnosis in repairing faulty knowledge engineered AN content, as well as the significant advantage of structured knowledge that reflects domain structure vs. flat representations in terms of learning speed. Of course, as problem size increases, the benefit of knowledge structure becomes more apparent, as can be seen in these results. This benefit is due to the restriction

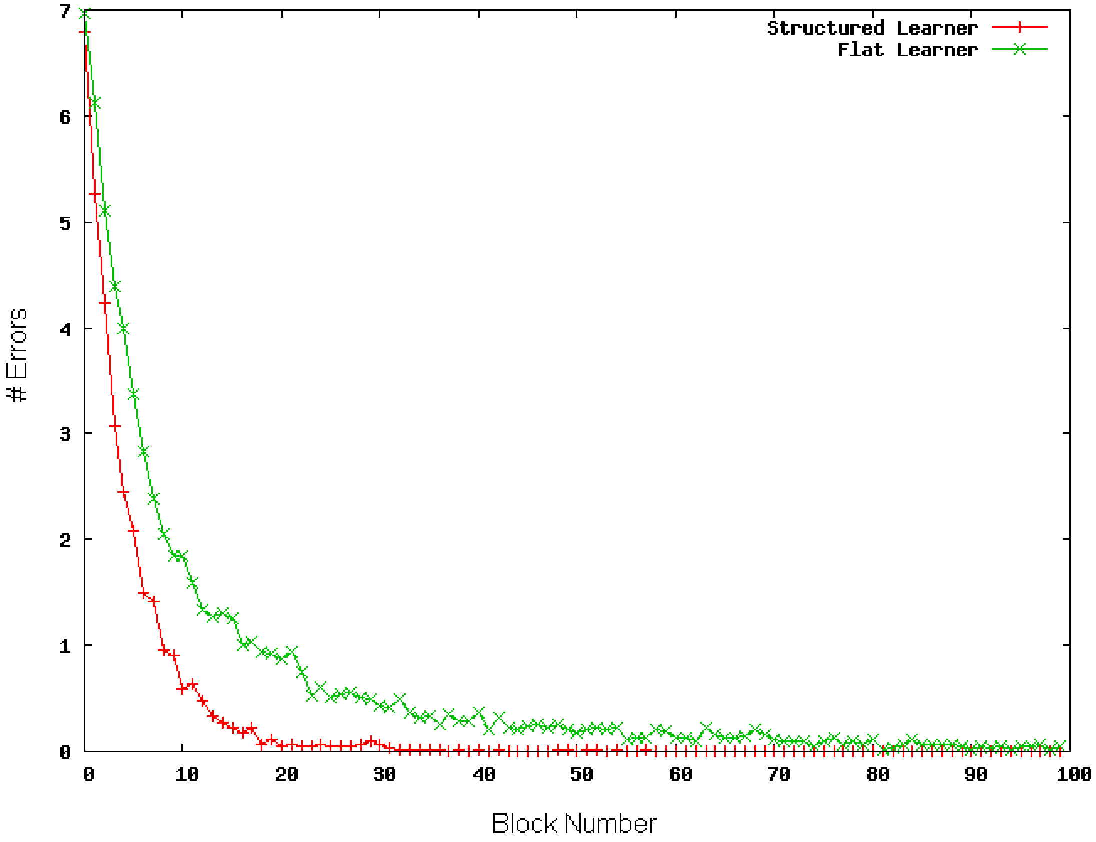


Figure 8: Per-block error rates of AN-rote learners vs. unaugmented rote learners for layer sizes 4, 2, 1.

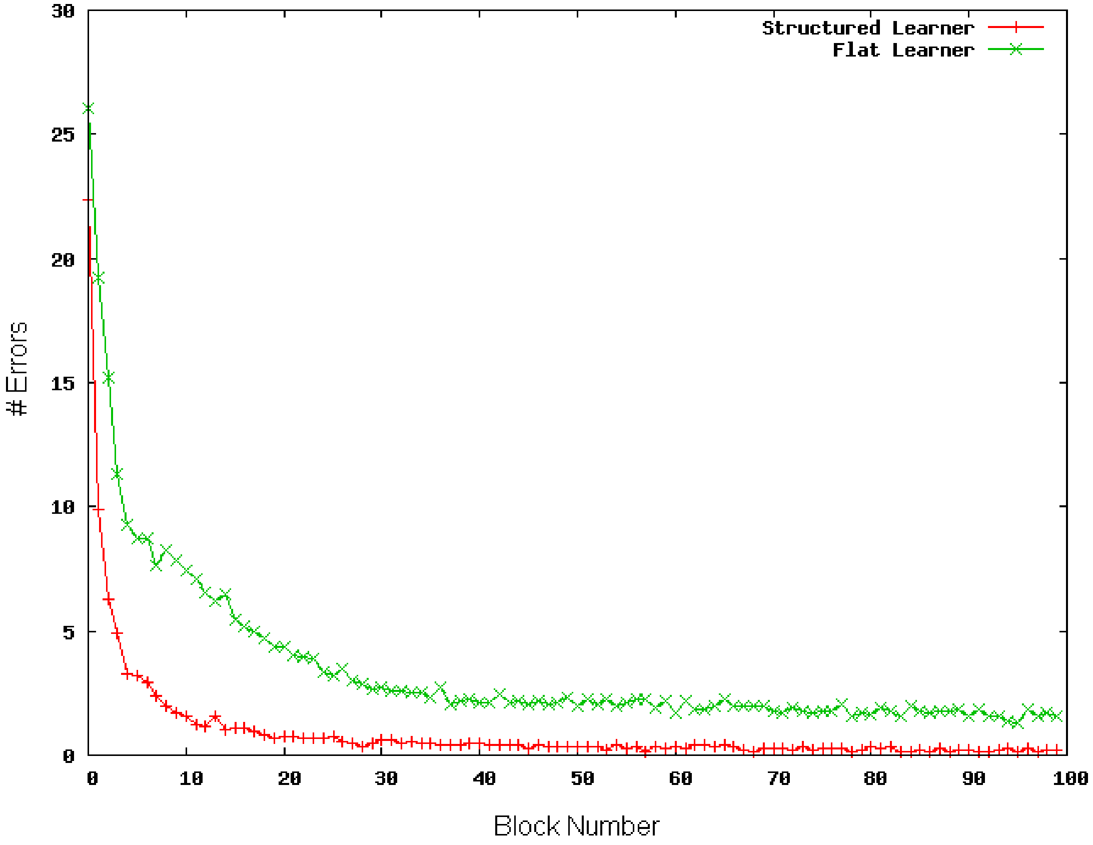


Figure 9: Per-block error rates of AN-rote learners vs. unaugmented rote learners for Layer sizes 8, 4, 2, 1.

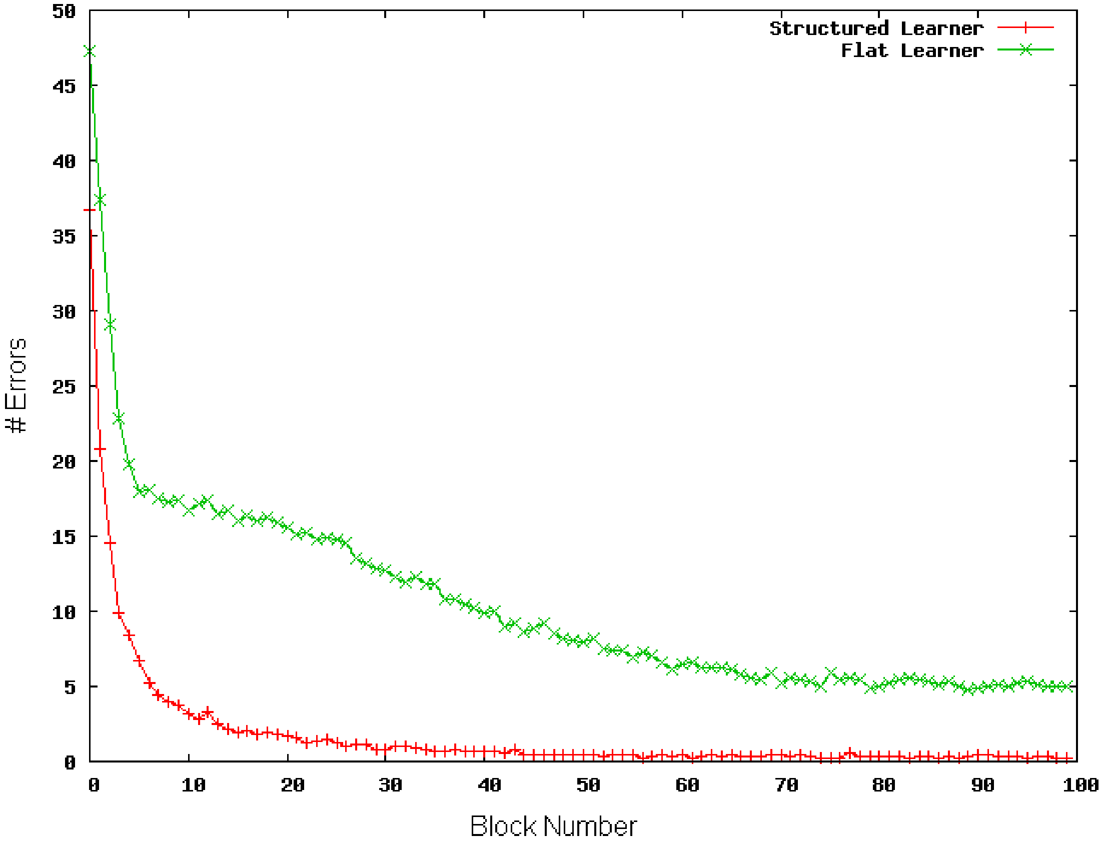


Figure 10: Per-block error rates of AN-rote learners vs. unaugmented rote learners for Layer sizes 16, 8, 4, 2, 1.

bias imposed on the hypothesis space available to the learner by the hierarchical knowledge structure and the semantic constraints encoded by EVPs. The nature of this restriction bias is discussed more formally in Section 7.3, and the benefit due to the semantic constraints afforded by EVPs is empirically demonstrated in Chapter 5.

4.1.2 Artificial Neural Networks

As indicated above, ANs do not commit to rote learners within nodes, but rather can make use of a variety of supervised classification learning techniques within nodes. In order to demonstrate the generality of ANs with respect to the classification learners used within nodes, this section describes results obtained after integrating the AN framework with artificial neural network (ANN) code provided by Tom Mitchell and his students. This integration allows us to replace the rote learners used within AN nodes in most of the experiments described here with ANNs. These results show, as expected, that an AN-ANN system has significant advantages over an ANN-only classifier.

We used a randomly generated set of synthetic learning problems to compare the performance of AN-ANNs with unaugmented ANNs. As in the synthetic experiments described previously, the environment consists of a fixed Abstraction Network, over which no learning will occur, that represents the correct, target content (and structure) for the problem. Given this fixed AN, we then again create a separate learner AN, with an ANN inside each node, that will be initialized with random knowledge content and be expected to learn to functionally match the content of the target AN. We also create a randomly initialized unaugmented ANN that will be used to learn the same classification task. All ANNs, whether within the AN structure or operating in isolation, used the same backpropagation algorithm for learning. For these experiments, learning rate was fixed at 0.3, momentum was fixed at 0.3, input layers contain one node per input, output layers contain one node per possible output value, and hidden layers contain a number of nodes equal to 3 times the number of nodes in the input layer. As before, because the work described here is concerned only with repairing content and not structure, we build the AN-ANN learner with correct structure that matches that of the fixed AN. Departing from the other experiments, in these experiments we first generate training and test sets. For every example that will be part of either the fixed training set or fixed test set, we generate a pseudo-random sequence of floating point numbers to serve as input values. Next, we repeat the following procedure, one repetition of which we call an *epoch*:

1. For each example in the training set:
   (a) Perform inference with the fixed AN, saving the output values of all intermediate nodes and the root.
   (b) Train both the AN-ANN and the unaugmented ANN systems based on the preceding substep's inference over the fixed AN. In these experiments we do not use the self-diagnosis procedure described in Section 2.4, but instead execute every EVP in the learner AN for every training example, and train the associated learner whether the value produced was correct or incorrect. This procedure ensures a stable distribution of training examples for ANNs within each AN node, while still depending crucially upon the availability of EVPs at each AN node.
2. For each example in the test set:
   (a) Perform inference with the fixed AN, noting the value produced at the root.
   (b) Perform inference with both the AN-ANN and unaugmented ANN systems, and determine whether the top-level values produced match that produced

by the fixed AN. If the value produced by a given learner does not match that of the fixed AN, count an error for that learner.

As before, EVPs within the inputs of both ANs are set up to quantize the floating point observations, and these quantized values also form the inputs to the unaugmented ANN. EVPs are not needed at non-leaf nodes in the fixed AN, since no learning will occur. EVPs at non-leaf nodes in the learning AN are set up to examine the saved output value from the corresponding node in the fixed AN, while the output value from the root of the fixed AN is all that is needed to train the unaugmented ANN. In these experiments we again use randomly-initialized table based rote "learners" within each node in the *fixed* AN, to simply provide a randomized mapping from inputs to outputs (that is, we simply use these as fixed tables, and not really as learners). Results obtained in three representative experiments are depicted in Figures 11-13. In these experiments, we again use ANs with a binary tree structure, with varying layer sizes – either 8-4-2-1 or 16-8-4-2-1. We also varied the number of choices that could be produced by each node, using either 3 or 4 values per node. For the experiment shown in Figure 13, the training set contains 1,000 examples, while the test set contains 10,000 examples. For the experiments shown in Figures 11 and 12, both the training and test sets contained 1,000 examples. In each case, we ran the complete experiment 5 times (re-randomizing all learners and the fixed AN each time, etc.), and Figures 11-13 depict the average error values in each epoch across these runs.

Clearly, it appears that AN-ANNs have a distinct advantage in error decrease per example and in the final error achieved. Based on these results, it does appear, as expected, that the advantage of adding AN structure to an ANN-based solution to a classification problem grows as problem complexity increases.

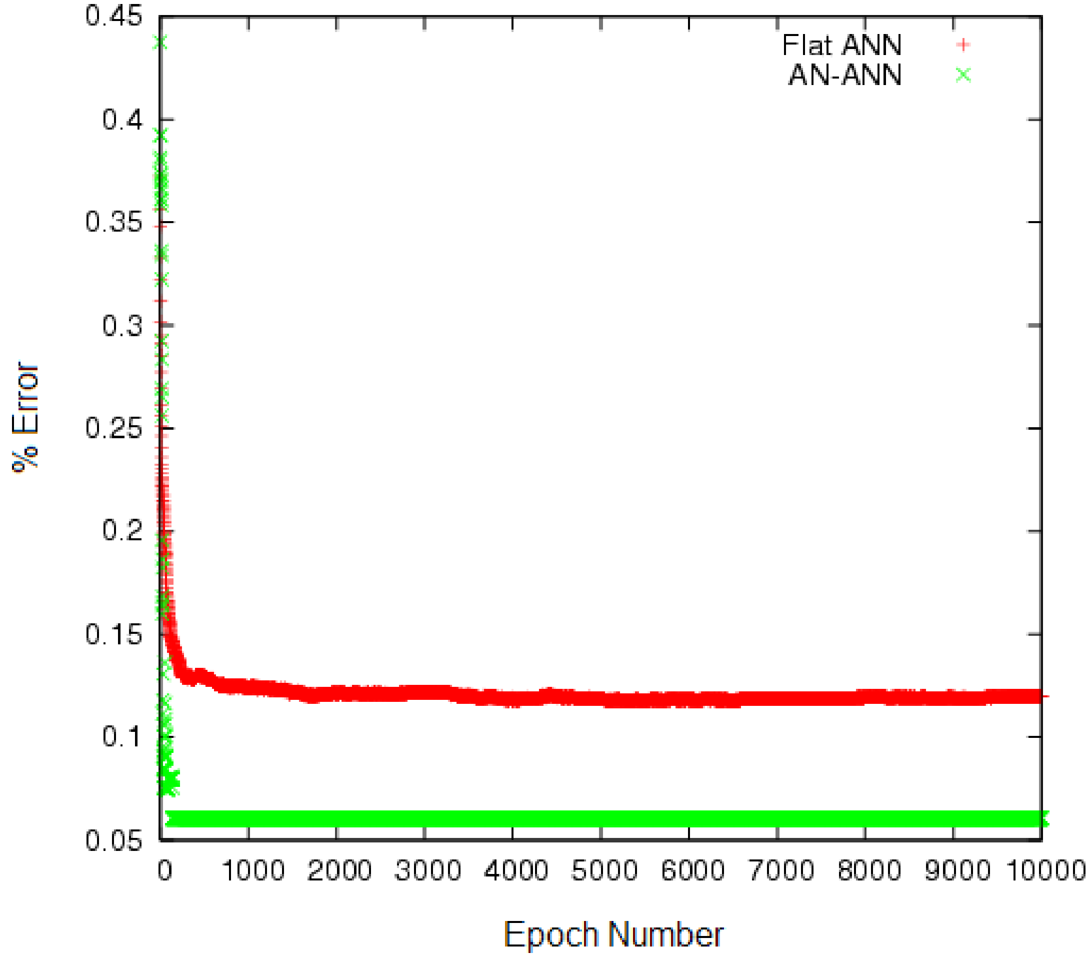


Figure 11: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 8-4-2-1, 3 choices per node.

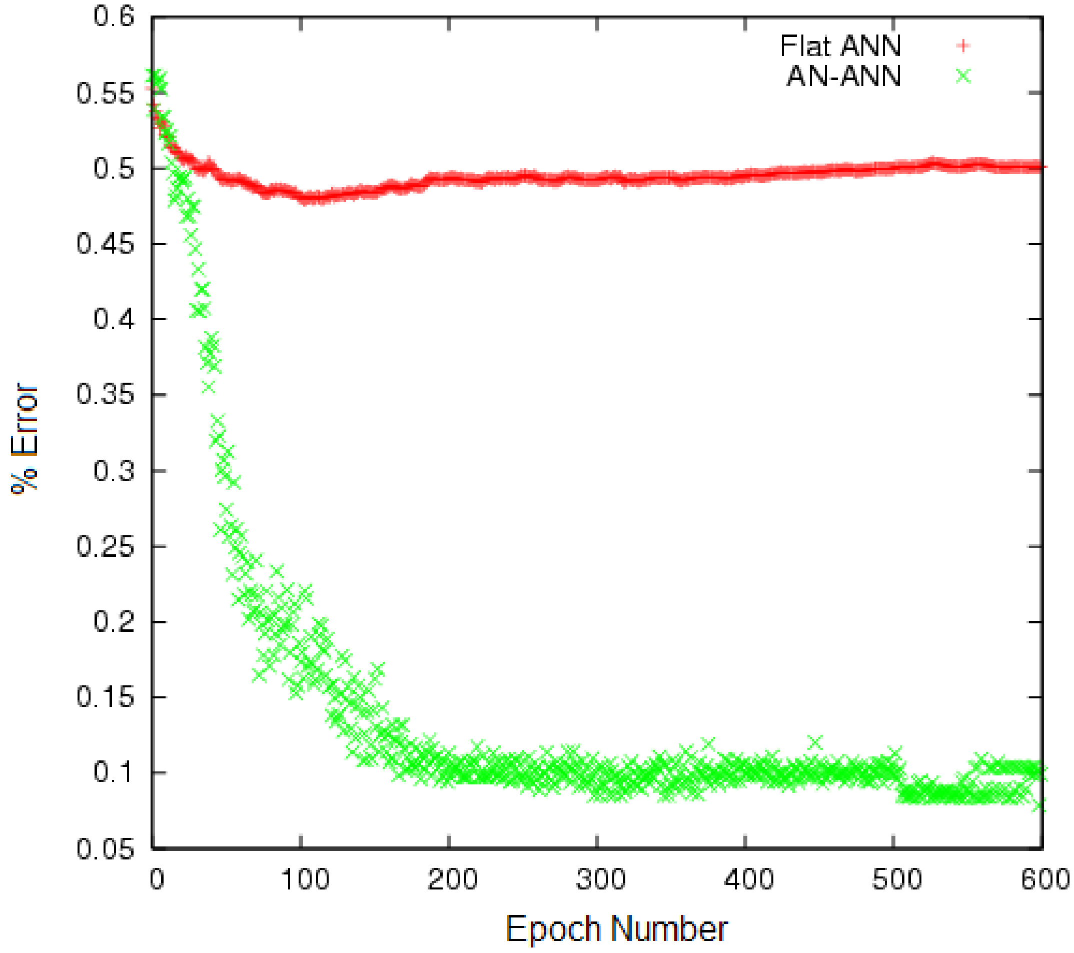


Figure 12: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 8-4-2-1, 4 choices per node.

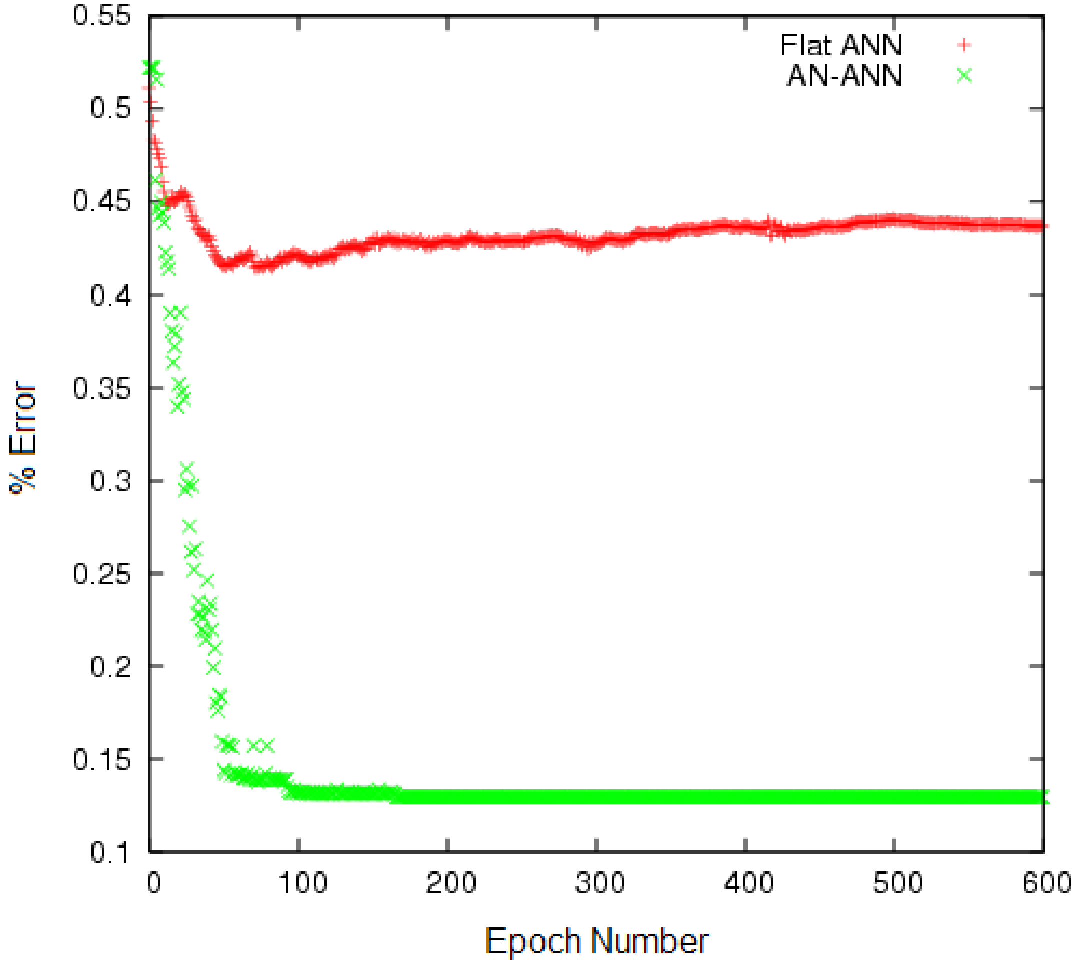


Figure 13: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 16-8-4-2-1, 3 choices per node.

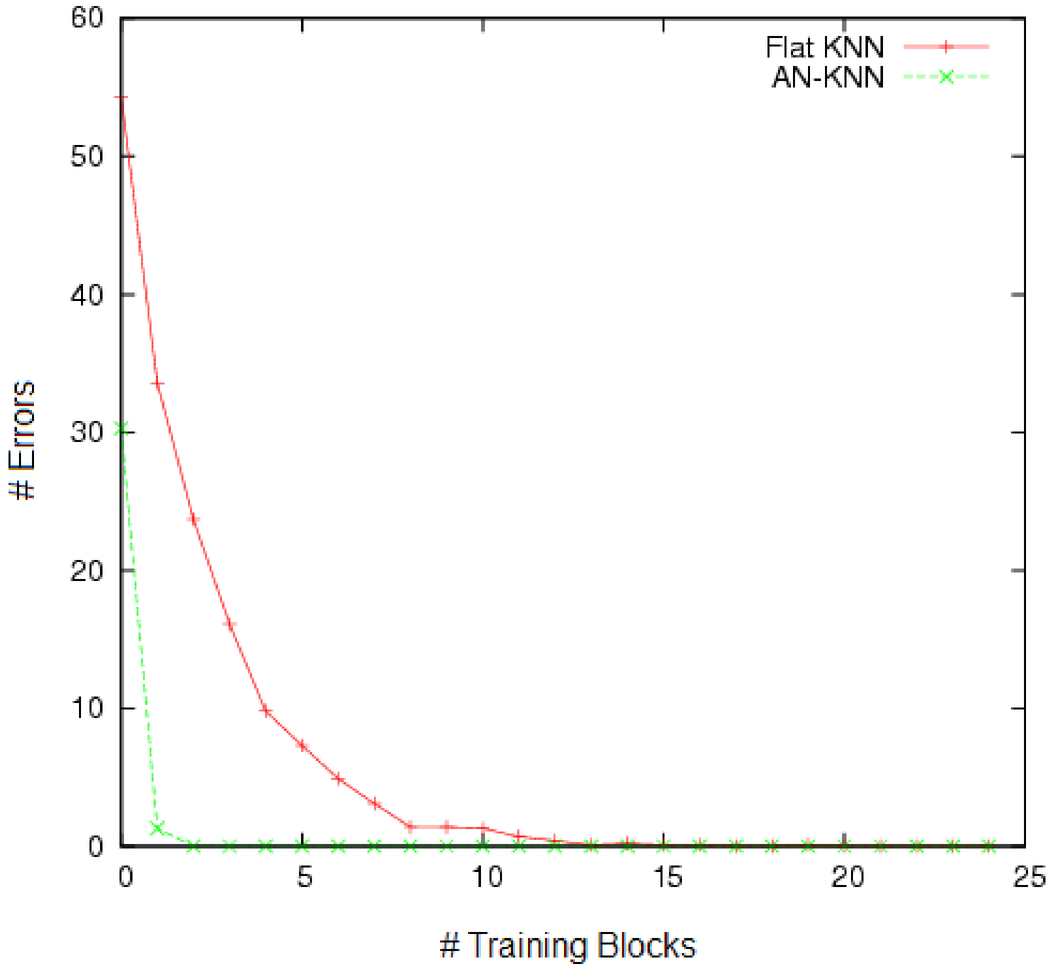


Figure 14: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 4-2-1, 4 choices per node.

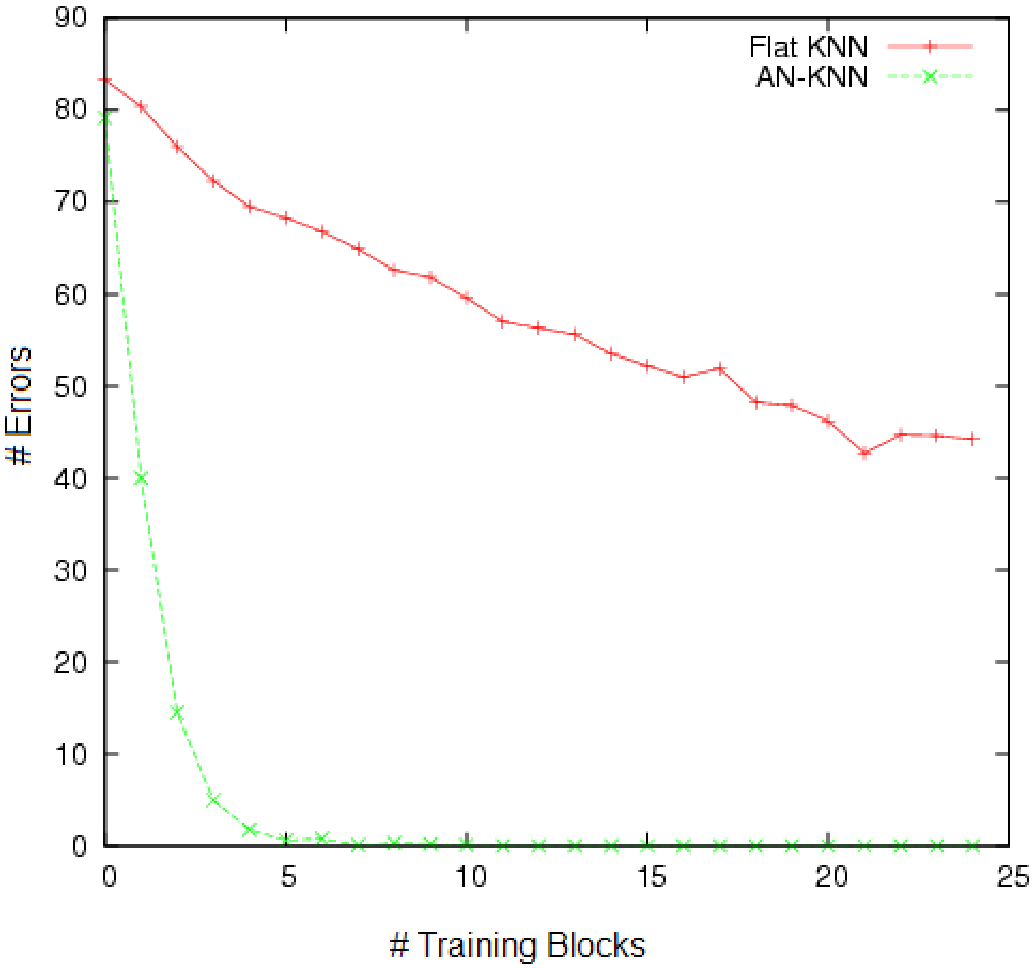


Figure 15: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 4-2-1, 8 choices per node.

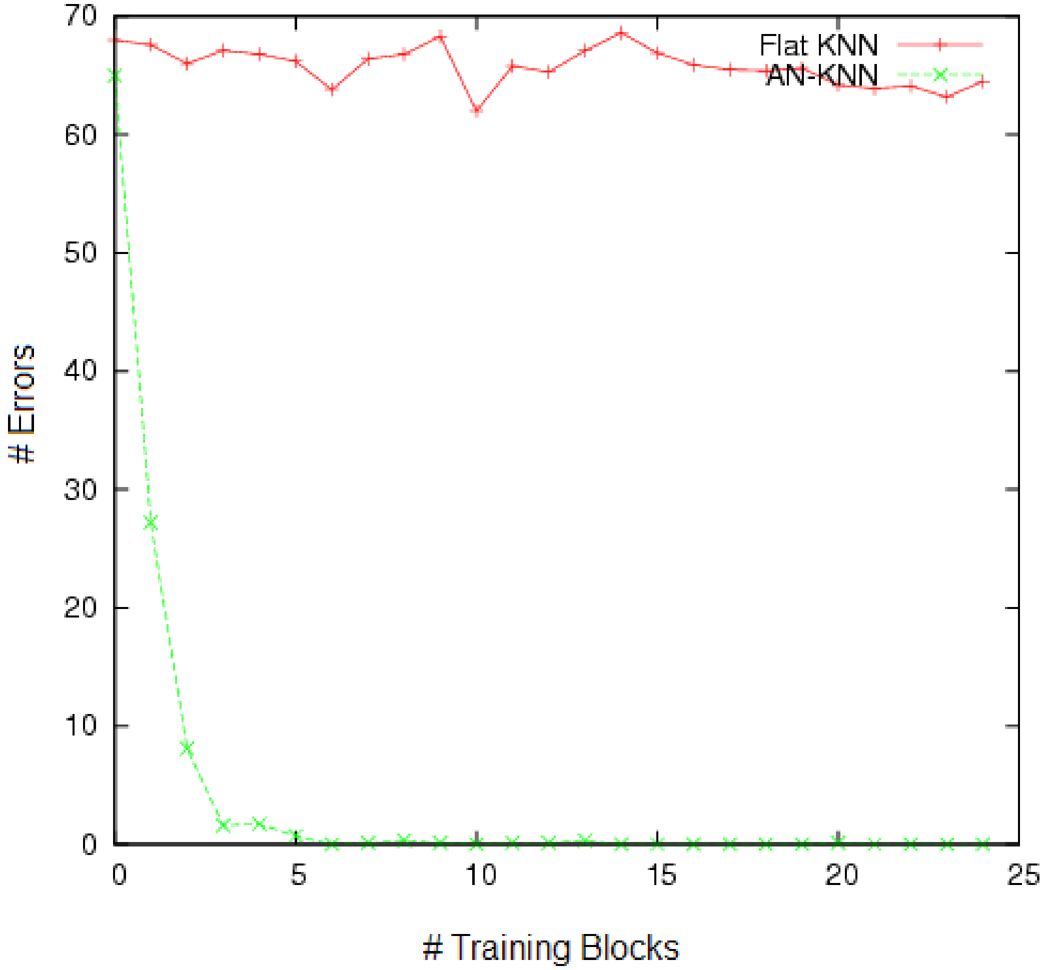


Figure 16: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 16-8-4-2-1, 4 choices per node.

4.1.3 k-Nearest Neighbor Learners

We also integrated ANs with kNN learners. As with the two previously integrated learner types, we performed experiments in a synthetic domain to gauge performance of kNN learners working in conjunction with the AN framework to unstructured (flat) kNN learners working on the same tasks. The experimental conditions for these experiments match those used for table-based rote learners. The 'k' parameter in these tests was set to 1. Results for problems of varying complexity are summarized in Figures 14-16, and are similar to those demonstrated for the two other learner types. As in past experiments, the difference between learners structured with the AN framework and unstructured learners increases with problem complexity, as expected.

4.2 Other Domains

4.2.1 FreeCiv City Location

In this section, the use of Abstraction Networks for the city resource production prediction problem first described in Chapter 1 is given in detail. Results of this experimentation are also given.

4.2.1.1 AN Representation

For the FreeCiv city resource production prediction task described at the beginning of Section 1.1, we use the structured matcher depicted in Figure 5, producing predictive classifications of map locations in a sequence of games. Within each node, we use a simple rote learner, defined in Section 3.1, with a threshold of 5.

4.2.1.2 Procedure

We have experimentally compared an AN-based learner using the network depicted in Figure 5 to a flat table-based rote learner. The goals of this experiment were to (1) determine the effectiveness of EVP-based metareasoning in increasing robustness in the face of faulty knowledge engineering and (2) to empirically illustrate the effect of hierarchicalization on learning speed outside of the synthetic domains already discussed. The effect of hierarchicalization on inference complexity is already well understood and is known to make inference significantly more manageable [37]. The flat learner consists of a single rote learner (rote learners are defined above) with an input formed from the outputs at all leaf nodes in the AN from Figure 5 and yielding the same output set as this AN. This output set contains three values, corresponding to poor, moderate and good resource production. These values indicate predictions about the resource production expected from a city built on a considered map location. Specifically, the values correspond to an expected degree and direction of deviation from a logarithmic baseline resource production function that was manually tuned to reflect roughly average city resource production. Each of the intermediate nodes in the AN has an output set consisting of 5 values in this experiment. The Empirical Verification Procedures are *quantizing EVPs*, described in Section 2.1, in that they simply check values in the game, such as the population growth of a city, and discretize the value into one of the 5 available output categories. The discretization functions were manually tuned in this experiment. The content of all involved table-based rote learners (those constituting the AN and the single one used for the flat learner) was initialized to zeros, which was known to be incorrect in some cases for each of the learners. All table-based rote learners used a learning threshold of 5. Because we expect resource production from cities built on various kinds of map locations to potentially differ qualitatively as games progress, we trained 3 AN-based learners and 3 flat rote learners, with one of each learning to make predictions about resource production in the early, middle or late stages of the game. Results reported are cumulative across all three learners of the appropriate type.

As in the non-batch experiments in the synthetic domain, here we train and evaluate the learners in an on-line, incremental fashion. Again, we evaluate prediction improvement during training by segmenting the sequence of examples into multi-example blocks, and comparing overall error rate between blocks. In this way, we are able to compare error rate around the beginning of a training sequence with the error rate around the end of that sequence.

Each turn of each game played is treated as a separate example. This means that an error is potentially counted on each turn of each game by producing a prediction based on the current state of knowledge, finishing the turn, perceiving the outcome of the turn, and then determining whether the value produced correctly reflects the resource production actually experienced on that turn. If the value is incorrect, an error is counted. Though as the game progresses, additional information becomes available, predictions are always made using only information available at the beginning of the game. Note that this error counting procedure contrasts with another possibility; producing a value only at the beginning of each game, and counting errors on each turn of the game based on this value, while continuing to learn on each turn. If the classification knowledge encoded by this FreeCiv domain AN were being used by a larger agent to actually play a game (e.g. the agent depicted in Figure 4), a classification produced by the structure would only be useful when the agent was deciding whether to place a city in a given location, and not after the city had already been placed. However, while the alternative of classifying only at the beginning of the game, before the city is built, more closely matches the intended *use* of the learned knowledge within a larger agent, we chose to instead allow a value to be produced on each turn in order to reflect the evolving state of knowledge as closely as possible in the error count. A negative consequence of this choice is that some overfitting within games may be reflected in the error count. However, a decrease in error rate between the first and last block in a sequence can be seen as evidence of true learning (vs. overfitting), since any advantage due to overfitting will be as pronounced in the first block of games as in the last.

In each trial, a sequence of games is run, and learning and evaluation occurs online as described above. The AN-based learner is trained on sequences of 175 games, while the flat rote learner is allowed to train on sequences of 525 games. We trained the flat rote learner on sequences three times longer than those provided to the AN learner to determine whether the flat rote learner's performance would approach that of the AN learner over a longer training sequence. As described above, we segment these sequences of games into multi-game blocks for the purpose of evaluation; the block size used is 7 games. Each game played used a (potentially) different randomly generated map, with no opponents. The agent always builds a city on the first occupied square, after making an estimate of the square's quality. Building in the first randomly generated occupied square ensures that the learners will have opportunities to acquire knowledge in a variety of states. In order to compensate for variation due to randomness in starting position and game evolution, results are averaged over multiple independent trial sequences. Each result for the AN learner is an average of 60 independent trials. Each result for the flat rote learner is an average over 25 independent trials; each trial is time consuming, as each trial for the flat rote learner is three times as long as for the AN-learner, and it did not seem likely that further trials with the flat rote learner would offer significantly more information.

To compare the speed with which learning occurs in the two agents, we ran two separate sets of trials. The first set of trials was run in an environment where no city improvements were constructed in the area surrounding the city. The second set of trials did allow for the construction of city improvements, but had an identical environment in all other ways. For each set of environmental conditions, we measure the quality of learning by comparing the average number of errors counted in the first block of the sequences to the number of errors counted in the last block. In the case of the flat table learner, we make two comparisons. The first compares error in the first block to the block containing the 175th game, illustrating decrease in error over

Table 3: Average percent decrease (or increase, shown in parentheses) in error for decomposition-based learning implementation from block 1 to 7, and for the flat table learner from block 1 to blocks 7 and 21.

	AN learner	Flat Table Learner	
	7th block	7th block	21st block
Without city improvements	24%	(4%)	1%
With city improvements	29%	7%	10%

the same sequence length provided to the AN learner. We also compare error in the first block to error in the last block of the flat table learner's sequences, to determine whether the flat table learner's improvement will approach that of the AN learner over sequences three times as long. We perform this evaluation separately for each of the two environmental setups.

4.2.1.3 Results

The results of the experiment are summarized in Table 3 and are shown in detail for the AN learners across each block of games in Figure 17. The AN-based learner is able to produce a greater improvement in error rate in each case, as compared to the flat table learner, both after the same number of games and after the flat table learner has played three times as many games. For the two scenarios, the average improvement in error rate is 26.5% for the AN-based learners, compared to only 1.5% after the same number of training examples for the flat learner. The decrease in error across a typical sequence was not strictly monotonic, but did exhibit progressive decrease rather than wild fluctuation. Even after three times as many games had been played by the flat table learner, the decrease in error rate is significantly less than that achieved using ANs after only seven blocks. In one case, it appears that learning has not yielded an advantage in error rate in the flat table learner even after 525 games. Examining the complete set of results for intervening blocks does mitigate this impression to some

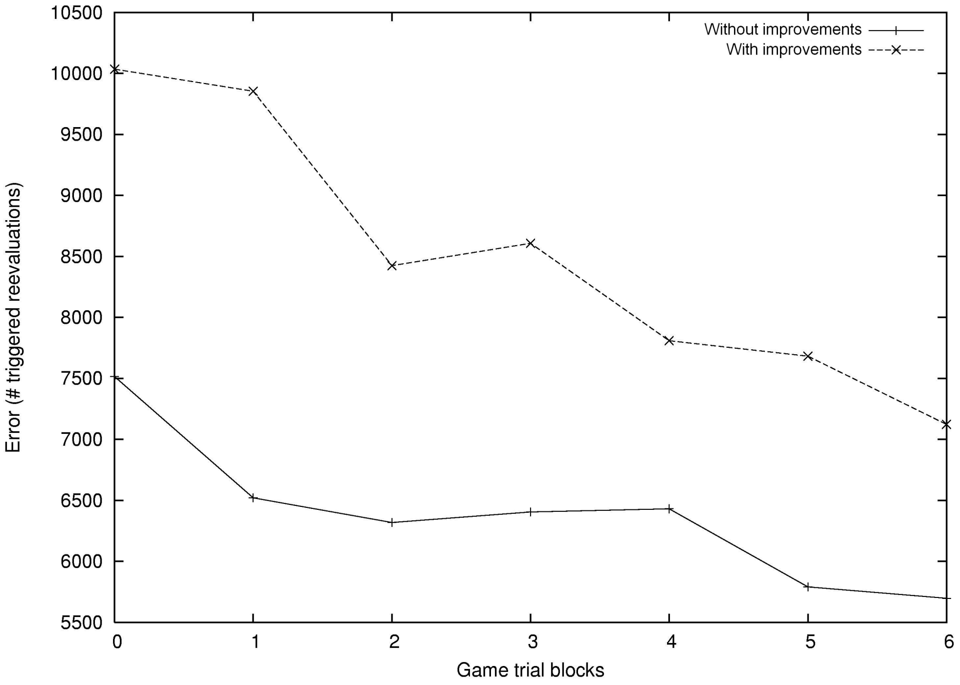


Figure 17: Average error rates by block in each FreeCiv trial.

extent, as an overall downward trend is observed, with some fluctuations. However, given that, for the flat learner, the fluctuations can be of greater magnitude than the decrease in error, the learning that has been achieved after this number of games does not appear significant. Based on the significant difference in observed learning rate, these trials provide evidence that the composite structure of ANs allow learning to occur more quickly in a large state space than is possible with a flat knowledge representation. Because the AN-based learners are able to improve their performance over time, it also appears that again, as in the synthetic experiments, EVP-based self-diagnosis and learning is effective in repairing content deficiencies in hierarchical classification knowledge.

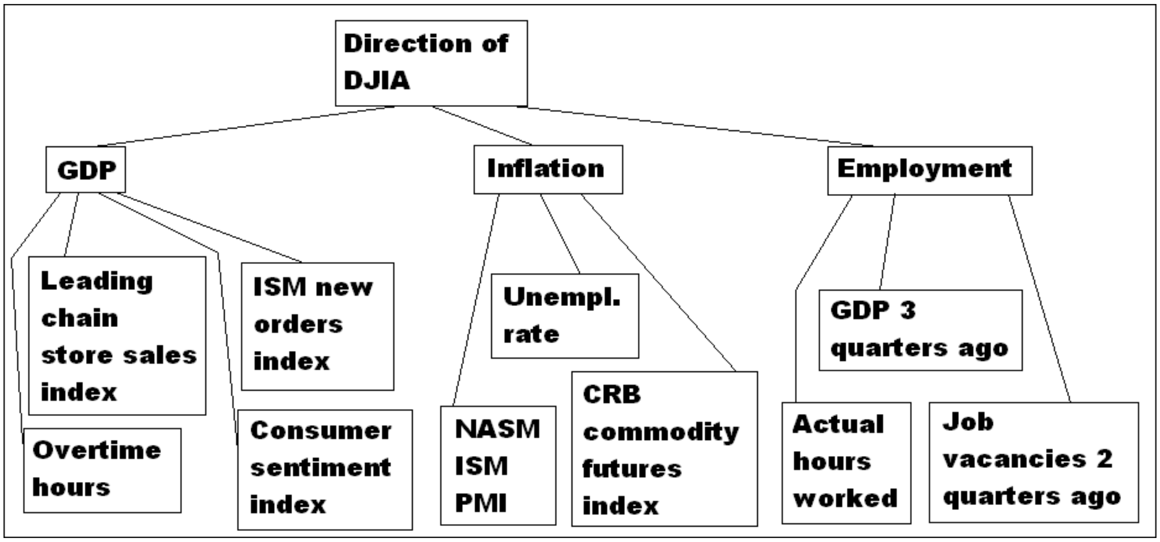


Figure 18: DJIA Abstraction Network

4.2.2 Dow Jones Industrial Average Prediction

To demonstrate that neither the learning task nor the learning method is restricted to the FreeCiv game, we will also describe results in a different domain in the economic arena. In this domain, we are interested in classifying the current economic status as described by various economic indicators (see Figure 18) into one of two classes: the Dow Jones Industrial Average (DJIA) will rise next month or DJIA will fall next month (these class labels form T). We chose the indicators and set up the structure shown in Figure 18 based on some studies of economic indicators [1][127][2]. S contains the values of these selected economic indicators. Some of the values can be obtained before classification; these values come from the current or past months. However, some of these variables represent future values that cannot be observed at classification time, but must be inferred along with the class label. The same special conditions regarding experimentation cost that were described for FreeCiv also hold here. All leaves in Figure 18 can be observed before classification, while the remainder are future values at classification time, available only in retrospect.

We used data from Jan 1960 - Nov 2005, yielding a total of 497 training examples.

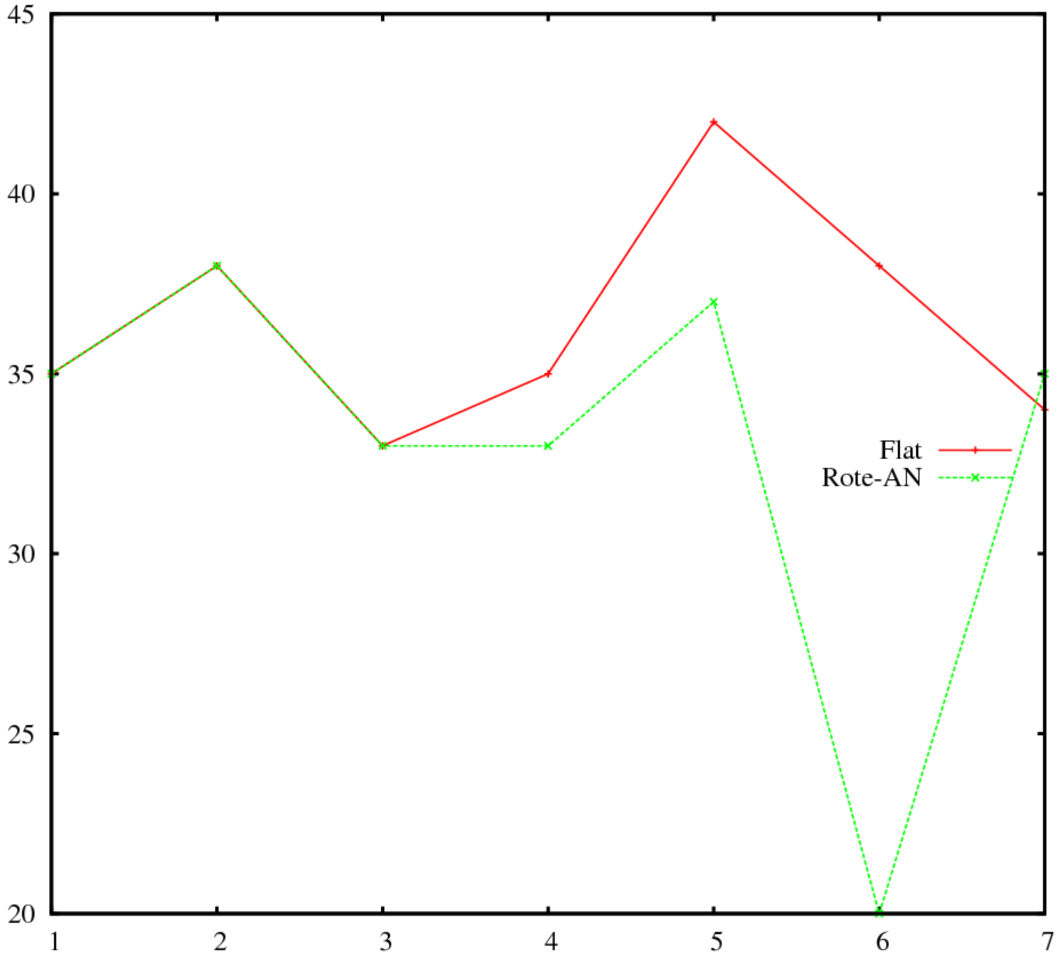


Figure 19: Average error rates by block in each DJIA trial.

As in FreeCiv, in these experiments rote learners were used within each node in the network. We manually tuned the number of output classes available to each node, based on observations of learning behavior. Again, each entry in each rote learner was initialized to zero. We observed a 23.4% decrease in error, comparing blocks consisting of the first 213 and the last 213 examples. The error rates for blocks sized 71 examples are depicted in Figure 19, which also includes data for a flat learner. This experiment helps to show that there is some more general applicability of EVP-based AN learning beyond the FreeCiv problem in the context of which it was initially tested.

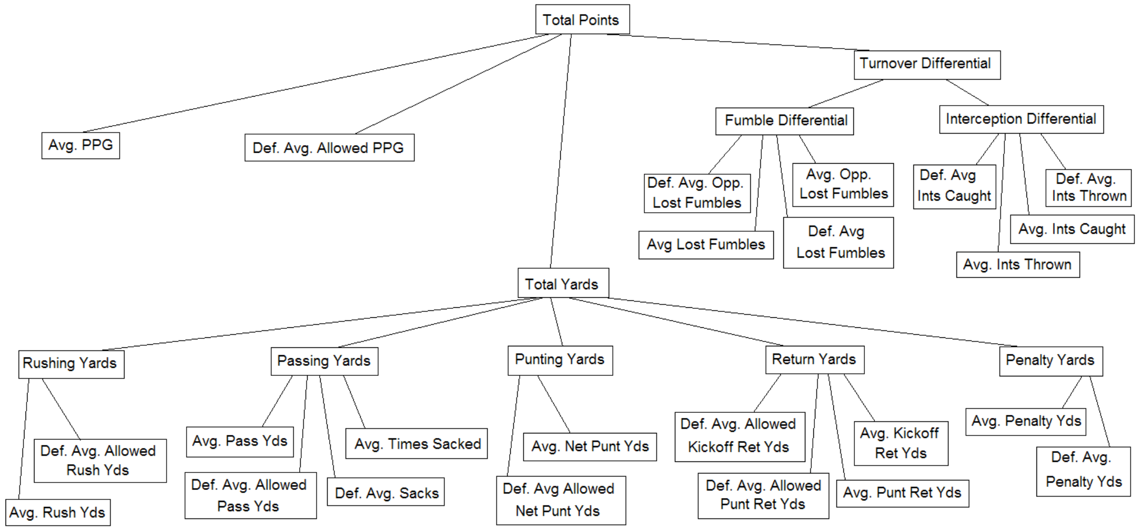


Figure 20: AN structure used in NFL prediction problem.

4.2.3 Football Prediction

We have applied EVP-based AN learning with both rote table learners and kNN learners to the problem of predicting the outcome (final score) of an NFL game. The AN learner depicted in Figure 20 was used in these experiments.

We used data for all games played in the 2006-2007 and 2007-2008 NFL seasons to train and test the learners, with the same online training/testing based strategy used in previous experiments. The results reported here are based on 50 separate random learning trials, each using exactly the same data but with randomized initialization of the learners' knowledge.

Results for these experiments are shown in Figure 21. Learner types shown include AN-kNN, AN-Rote and flat kNN. Flat rote learners could not be used because the memory requirements of the table were too large. It is interesting to note that the AN-Rote learner essentially fails to learn. It is likely that this is because of the size (input dimension) of the learning problem. Even with the additional bias afforded by the AN knowledge structure and the associated EVPs, it appears that this problem (or at least, this framing of this problem) is complex enough, and examples limited

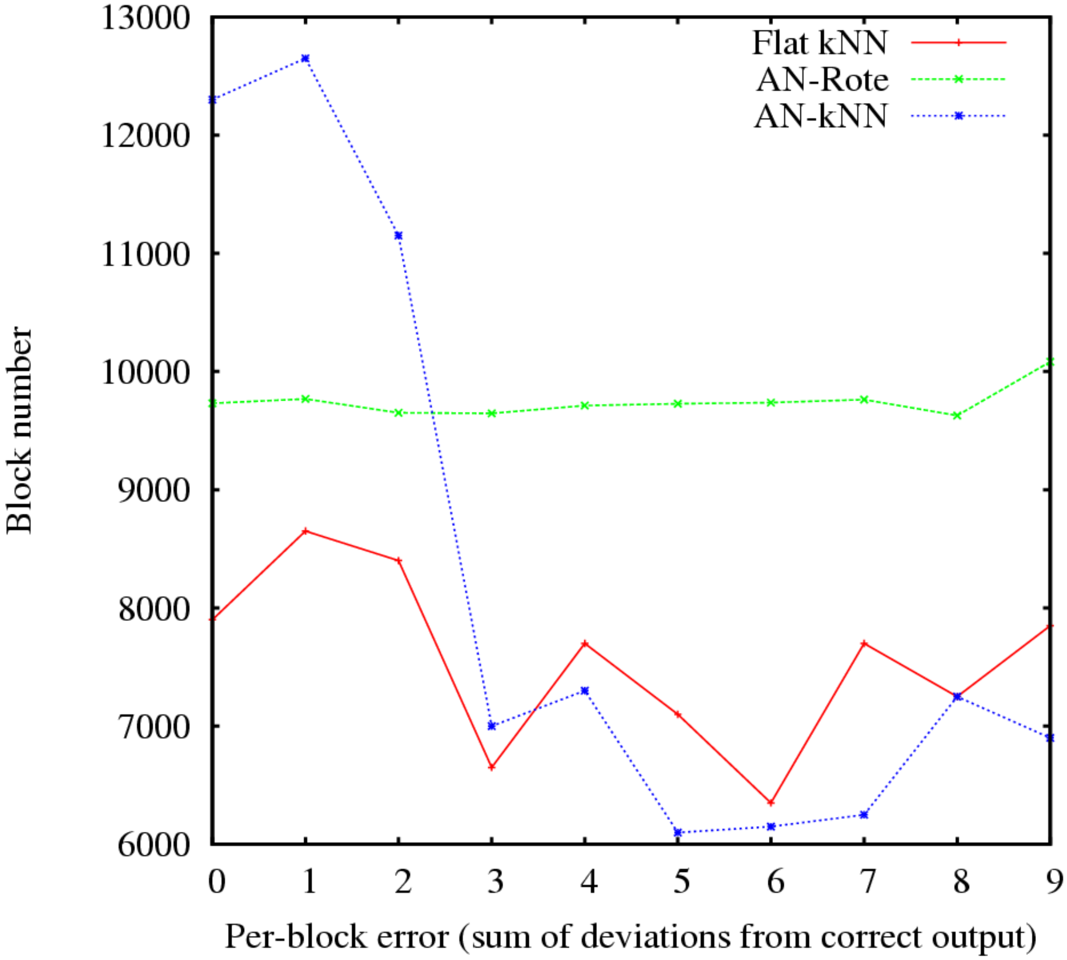


Figure 21: Per-block error vs. block number for various learner types.

enough, to require some inductive bias within the learners at individual nodes. Thus, this set of experiments demonstrates the importance of selecting intra-node learners with appropriate characteristics (e.g. bias) for a given application of ANs. After an initial lag, the AN-kNN learner matches or exceeds the performance of the flat kNN learner. This may be because, as a result of the non-exhaustive diagnostic procedure of Table 2, learning at the higher levels of the AN hierarchy depends on learning at the lower levels. But it may also reflect the fact that flat kNN learning is very fast. Indeed, it is often very difficult to improve upon kNN. The fact that AN-kNN beats flat kNN demonstrates the power of the abstraction hierarchy.

Further, the basic claim that knowledge repair is supported by EVP-based diagnosis and repair is supported by the decrease in error rate observed for the AN-kNN learner. However, it is unfortunately not clear that this domain, or either of the others, has so far provided a decisive and spectacular display of the advantages of AN technology in terms of reaching an extremely low final error rate. However, experiments in these domains have demonstrated the effectiveness of EVP-based diagnosis in allowing a metareasoning system to successfully repair knowledge stored in classification hierarchies and reduce error. Of course, in this case and in the cases of FreeCiv and DJIA prediction, it is highly likely that flaws in the knowledge engineering (KE) or gaps in available input features are responsible for failure to reach a lower final error. This issue is addressed more directly by work on faulty KE, which provides some evidence of the benefit of using ANs to structure classification learning even if KE is faulty. This work is discussed in the following section.

4.3 Effects of Degraded Knowledge Engineering

We have performed two sets of experiments in the synthetic domain of Section 4.1 dealing with the performance of AN learners when knowledge engineering is imperfect.

In all of these experiments, a binary AN hierarchy was used, with level sizes 16-8-4-2-1. We allowed each node in the hierarchy to produce 4 output values. Each non-leaf node contained a kNN learner with a k-value of 1. The results shown in this section are an average of 20 randomized trials, each consisting of sequences of randomly selected examples split into blocks of 100 for graphing purposes. In the first of these experiments, specific nodes are ablated from within the learner AN, connecting the child nodes of the removed node to the parent node of the removed node. In these experiments, no input information is lost through the node removals (inputs are never ablated), but we expect the hypothesis space restriction imposed by the AN structure to be diminished, and thus the efficiency of learning to decrease. This expectation is indeed borne out by the experiments, summarized in Figure 22. In these experiments, we still reach or approach zero error, as expected because the correct hypothesis is never eliminated from those expressible by an AN through this kind of ablation. However, the learning rate is negatively impacted as the restriction bias imposed by the AN is reduced. The keys for the graphs in this section refer to the location of nodes ablated by *level*. We consider leaf nodes to be level 0, the direct parents of leaf nodes to be level 1, etc. This notation is possible because of the balanced binary structure used in these experiments. An interesting note about these results is that, when ablating a single node, it appears to make no significant difference at which level of the hierarchy the node is removed. This suggests that impact on overall hypothesis space size is not dependent upon a concept's level of abstraction. We will return to this discussion in Section 7.3, where hypothesis space size is related to the information lost at nodes in an AN.

In the second set of experiments, whole subtrees beneath a selected node (or nodes) are pruned from the learner AN. This kind of ablation actually has the effect of *increasing* the restriction bias of the AN, as all hypotheses dependent upon the inputs beneath the ablated node are no longer expressible at all. This kind of removal

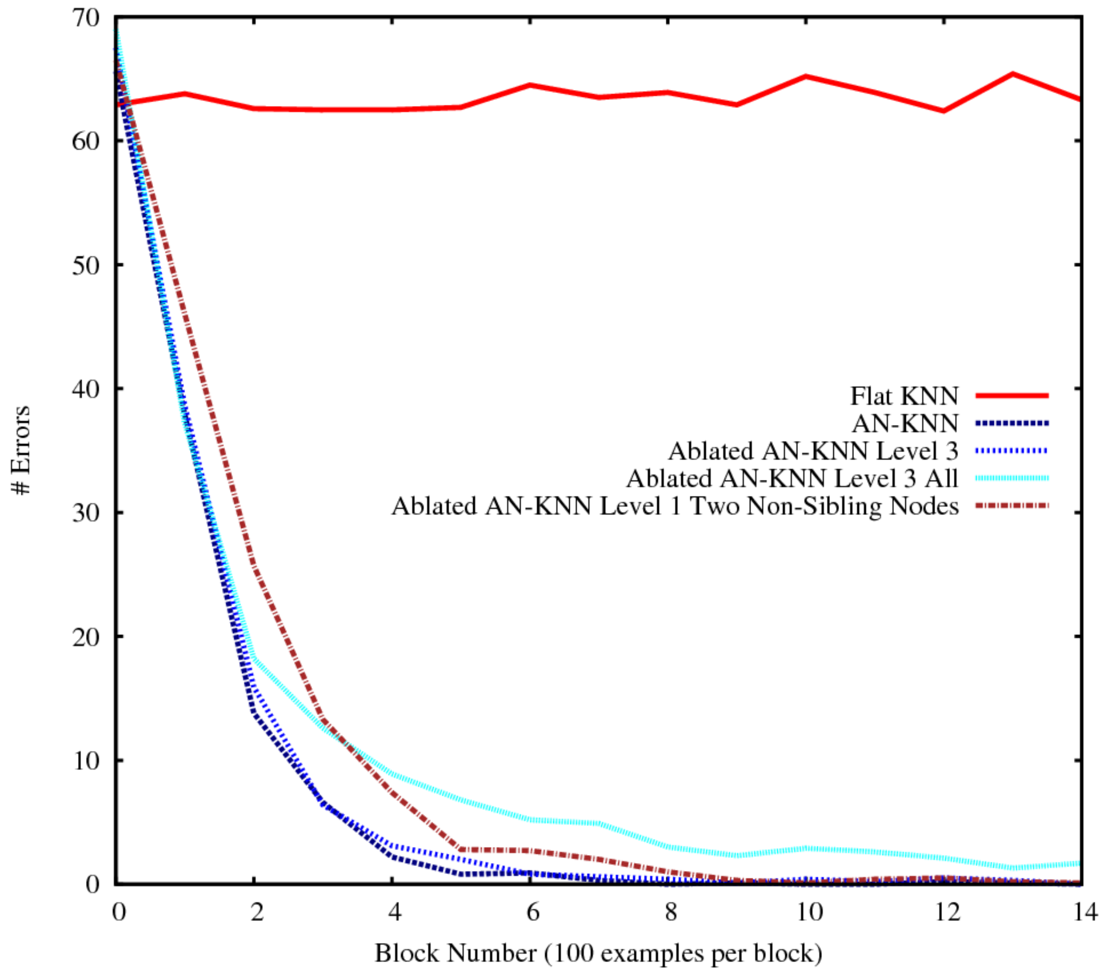


Figure 22: Results of ablating (groups of) individual nodes from an AN learner.

is equivalent to forcing *complete* information loss at the root of the ablated subtree. The problem here is that the restriction bias is likely to have now excluded the correct hypothesis, as inputs that may be needed for discrimination between two states could have been removed. These induced deficiencies are much more severe than those of the first set of experiments. As expected, the ability of the learner to correctly match the target function are more severely hampered, as illustrated in Figure 23. However, the final error reached is still below that of an unaugmented kNN learner after 1000 training examples – illustrating that, if *any* reliable structural information is available about a domain, there is substantial benefit to its exploitation if few training examples are available. Of course, over time the unaugmented kNN learner would reach zero error in this synthetic domain, once it has seen and memorized by rote each problem instance. However, in practical terms this situation would not arise. If it is known that some inputs are or may be pertinent, one can always feed them directly into the root node of an AN hierarchy even if intervening structure is not known. But it is interesting to note that in some sense, a designer is better off knowing about only a subset of the inputs relevant to a classification problem and having some good knowledge about an intervening abstraction structure than having full knowledge of the relevant inputs but no knowledge of the structure. While the latter scenario allows the designer to produce a learner that *theoretically* can express the correct hypothesis and thus would eventually reach zero error, in practical terms for large problems it will not be possible to gather enough training examples to get there. On the other hand while in the former scenario zero error will never be reached, some level of useful generalization can be made after relatively few input examples. In the trial where we ablated two non-sibling level 2 nodes, we have literally removed half of the problem inputs and still get a better error rate after 1000 examples have been seen!

The key finding in these experiments is that as knowledge engineering quality

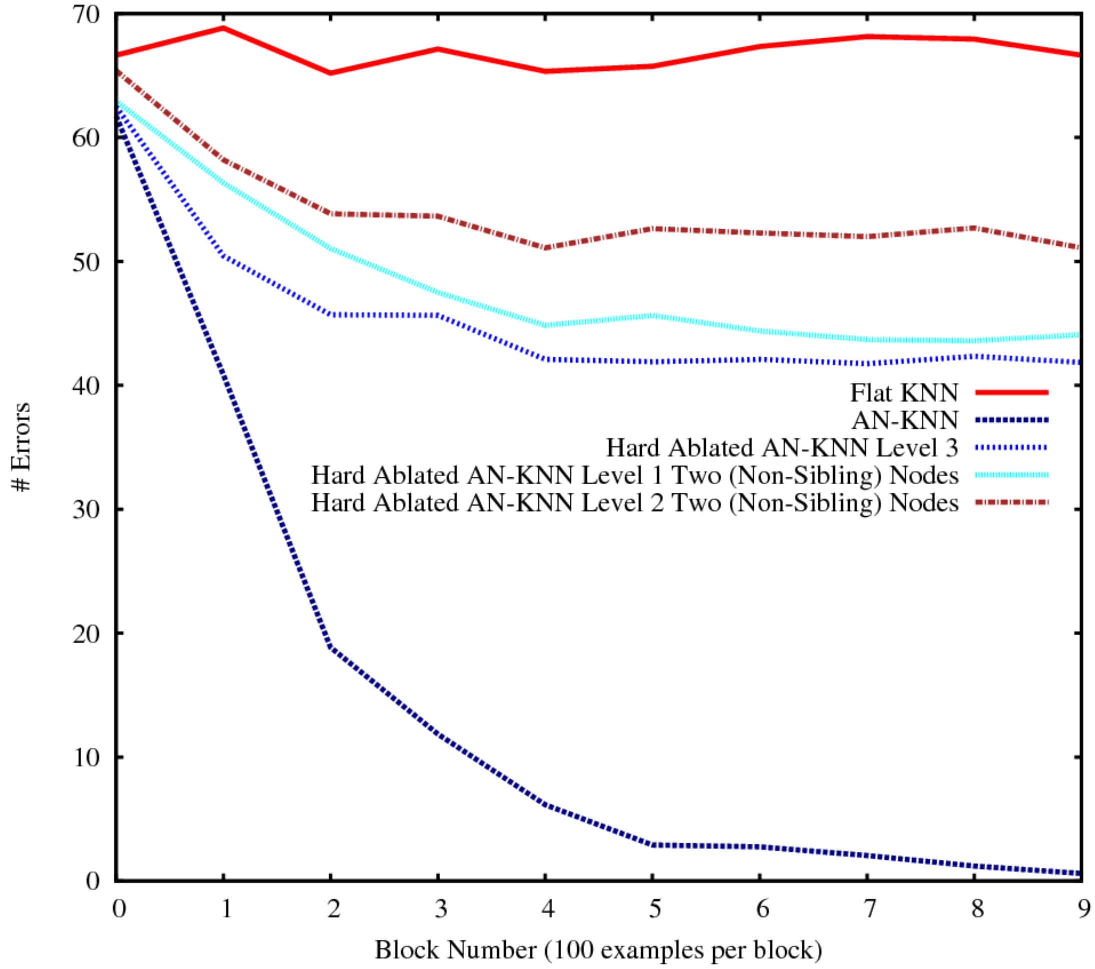


Figure 23: Results of ablating (groups of) subtrees from an AN learner.

degrades, there is a corresponding gradual degradation in the benefit obtained from using AN structure. Of course, here we have tested only two kinds of incorrectness in knowledge engineering. One could imagine many other kinds of errors, such as wiring nodes into the wrong location in an AN. In this case, one would expect the AN to learn to ignore information that is not pertinent to a particular classification. This would slow learning but should not impact final error beyond the effect of not having the information available in the correct location. Thus, the effect of such an error could be expected to be similar to that of ablating the subtree beneath the miswired node. In any case, it is not the intent of this thesis to experiment with, or even identify an exhaustive taxonomy of conceivable errors in knowledge engineering. However, this section does provide some sense of the kinds of degradation in learning rate (when intermediate abstractions are missed but all needed inputs are intact) and final error levels (when needed inputs are not present) that one can expect under two kinds of faulty knowledge engineering that seem likely to occur in practice when designing classification hierarchies.

CHAPTER V

LEARNING WITH UNSPECIFIED CONCEPT SEMANTICS

In the experiments described in this chapter, we again train ANs with perfectly designed structure within the synthetic domain described in Section 4.1. In these experiments, we sometimes remove the EVPs from some of the AN nodes, and train by executing standard error backpropagation [94] over those sections of the structure from which EVPs have been removed. We will refer to these ANs with some EVPs removed as *hybrid* learners, as they combine standard backpropagation with EVP-based diagnosis. The goal of these experiments is to demonstrate that the power of the semantic pinning of nodes provided by EVPs is substantial. That is, the generalization power of ANs is not solely due to the hierarchical structuring of the knowledge, but also derives in part from this semantic pinning.

5.1 Experimental Setup

In these experiments, we always used ANNs within all AN nodes. We also eliminate the quantization of outputs at nodes that do not have EVPs. This is necessary to cause those nodes to produce differentiable functions such that backpropagation can be used to push error back from ancestors into those nodes. We always used learner and generator ANs of four levels with binary tree structures (level sizes 8-4-2-1). In each case, we allow 3 values to be produced by each node in the hierarchy, including the root. We also trained a flat learner on the same problem, for the sake of providing a baseline for comparison. We have run experiments with one EVP, removed at the layer immediately above the leaves; two EVPs, removed at peer nodes within the layer immediately above the leaves; and three EVPs, removed in the form of a connected subtree rooted in the layer immediately beneath the root, thus also involving two peer nodes in the layer immediately above the leaves. For each of these learner setups, we experimented with various training set sizes – 60, 125, 250, 500 and 1000 examples. We always used a test set consisting of 1000 examples. In each case, the goal was to train until the error rate ceased to decrease, or until there was evidence of overfitting. Results reported here are the average of 5 independent trials, where learner knowledge and the training/test sets were re-randomized in each trial, in the same manner as in the experiments described in Section 4.1.2.

5.2 Results and Discussion

The results of the runs in a representative subset of these experimental setups are depicted in Figures 24-32 below, and the results of all runs are summarized in Figure 33.

The key result illustrated in Figures 24 through 32 is that in all cases, ANs that are missing EVPs from one or more nodes fail to reach a final error as low as the complete ANs. In general, this final error is higher (worse) when more EVPs are removed, and when the training set size is smaller. A progressive degeneration in learning ability as more EVPs are removed is apparent as a general trend, as demonstrated by Figure 33, which summarizes the results of all experiments run with hybrid ANs, showing the average final error rate for each experimental setup. It can also be seen from Figure 33 that larger training sets generally lead to lower final errors for each learner type, as one might expect. This main result demonstrates that part of the generalization power of ANs comes from the semantic pinning of nodes within the AN by the associated EVPs.

Somewhat surprisingly, in Figure 32, we see that in this case, the AN with 3 EVPs removed has failed to achieve a lower final error rate than the flat learner!

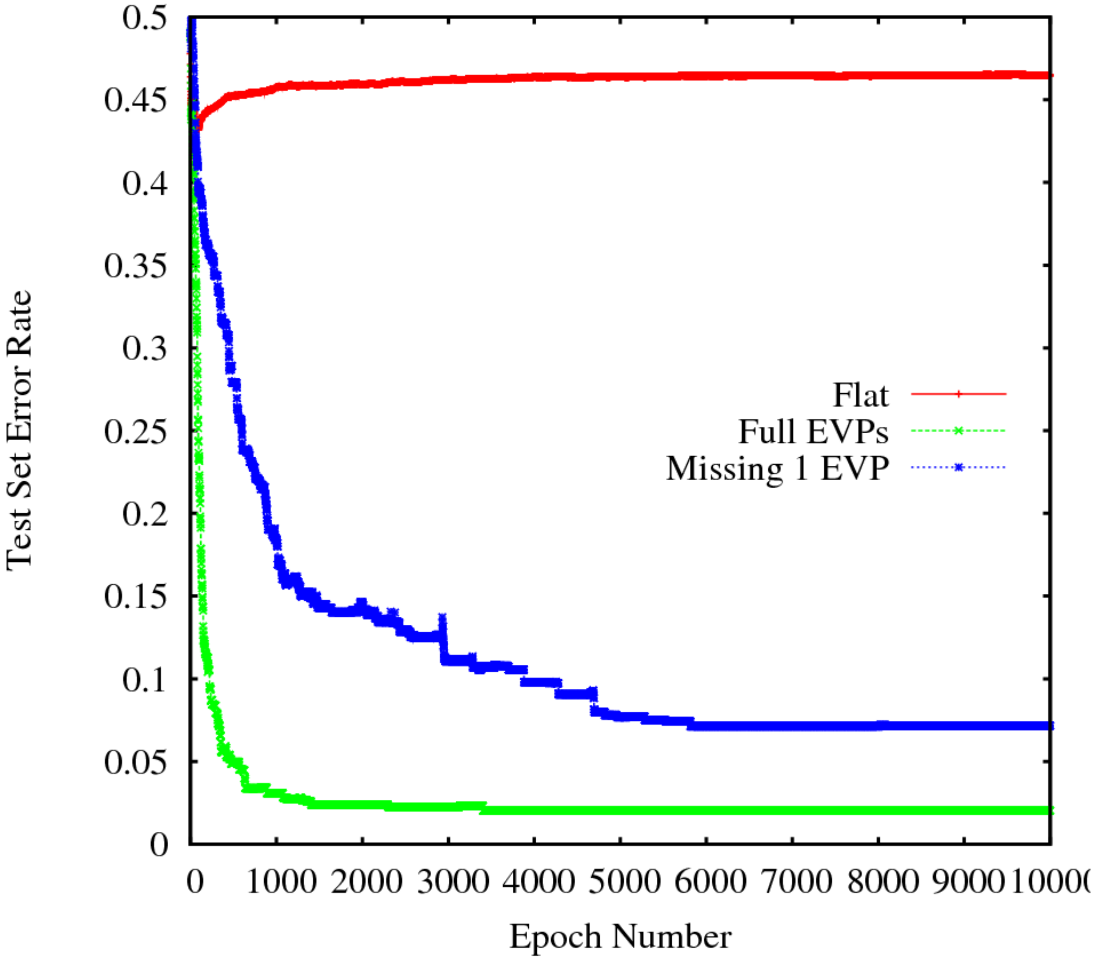


Figure 24: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 60 examples, where the hybrid learner is missing one EVP.

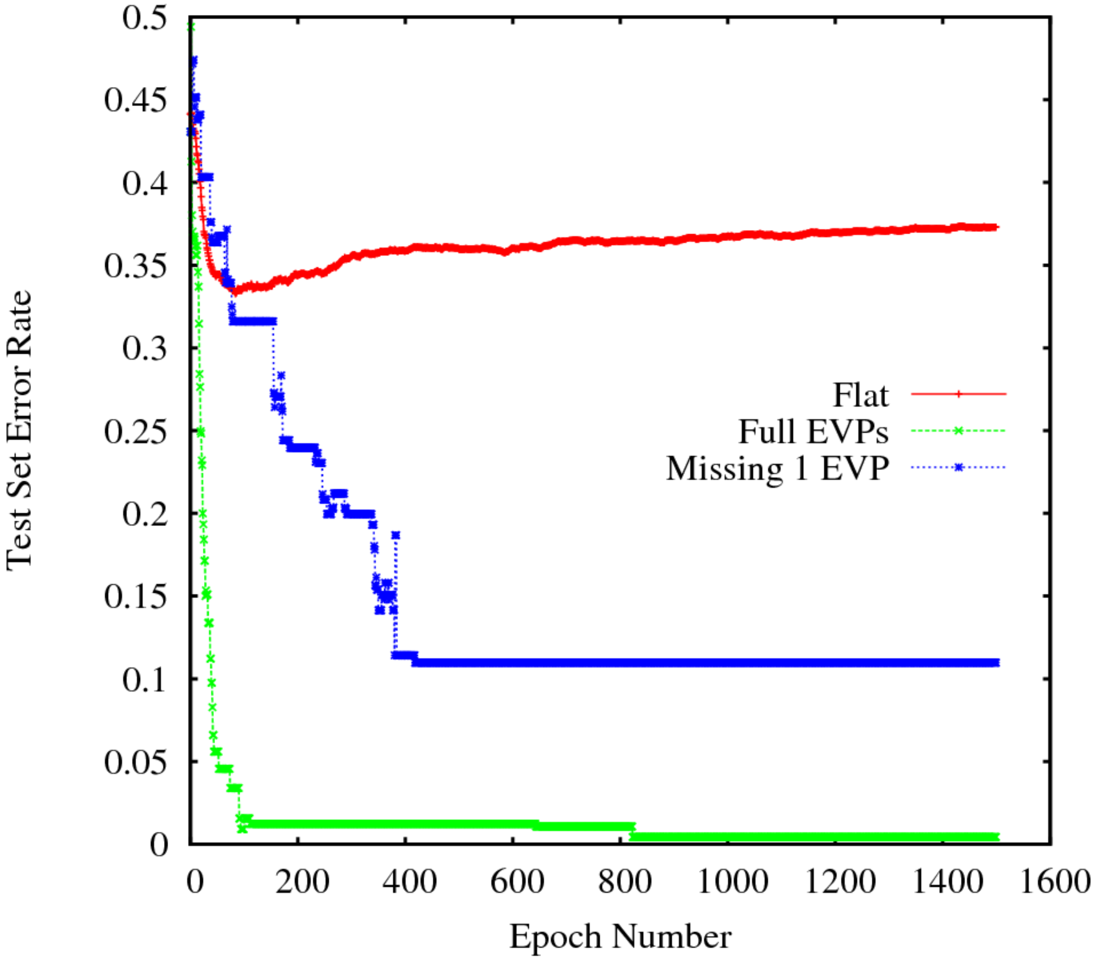


Figure 25: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 250 examples, where the hybrid learner is missing one EVP.

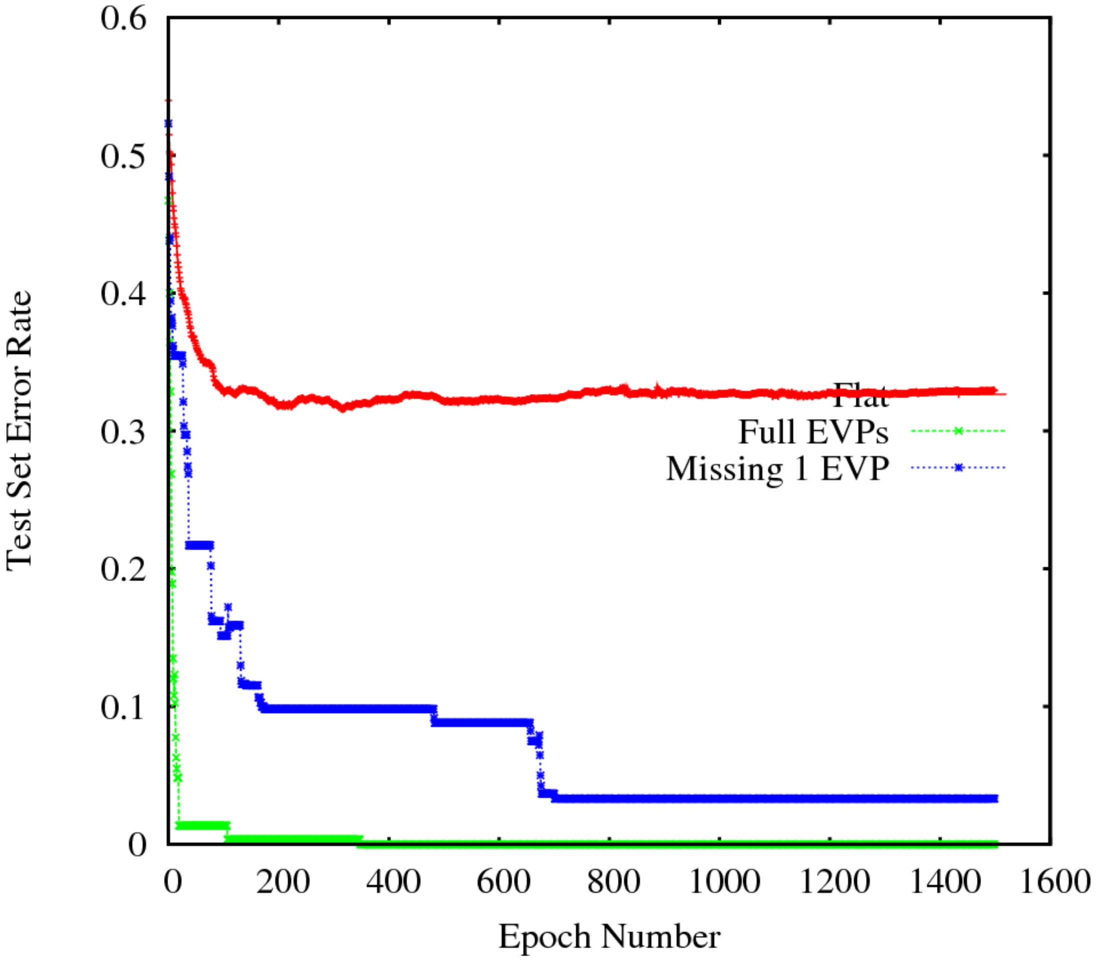


Figure 26: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 1000 examples, where the hybrid learner is missing one EVP.

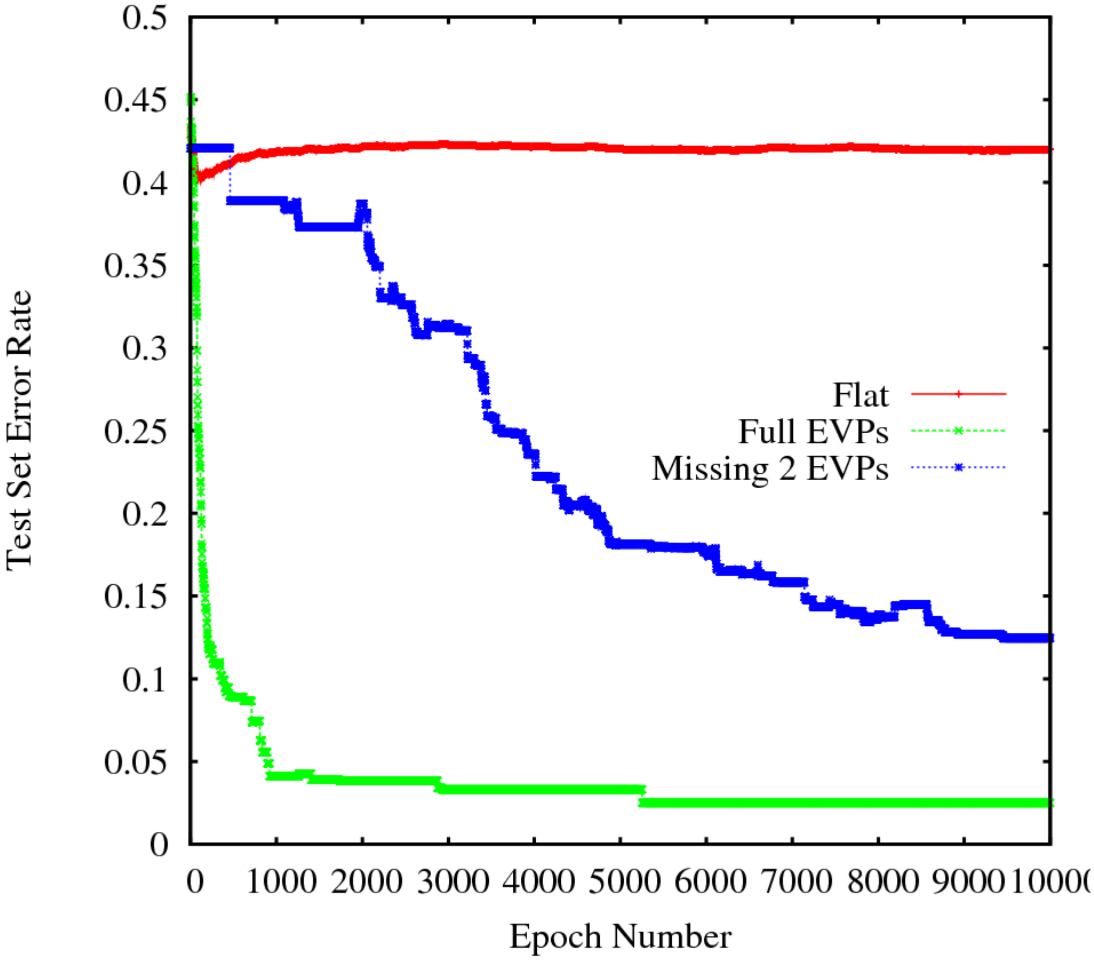


Figure 27: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 60 examples, where the hybrid learner is missing two EVPs.

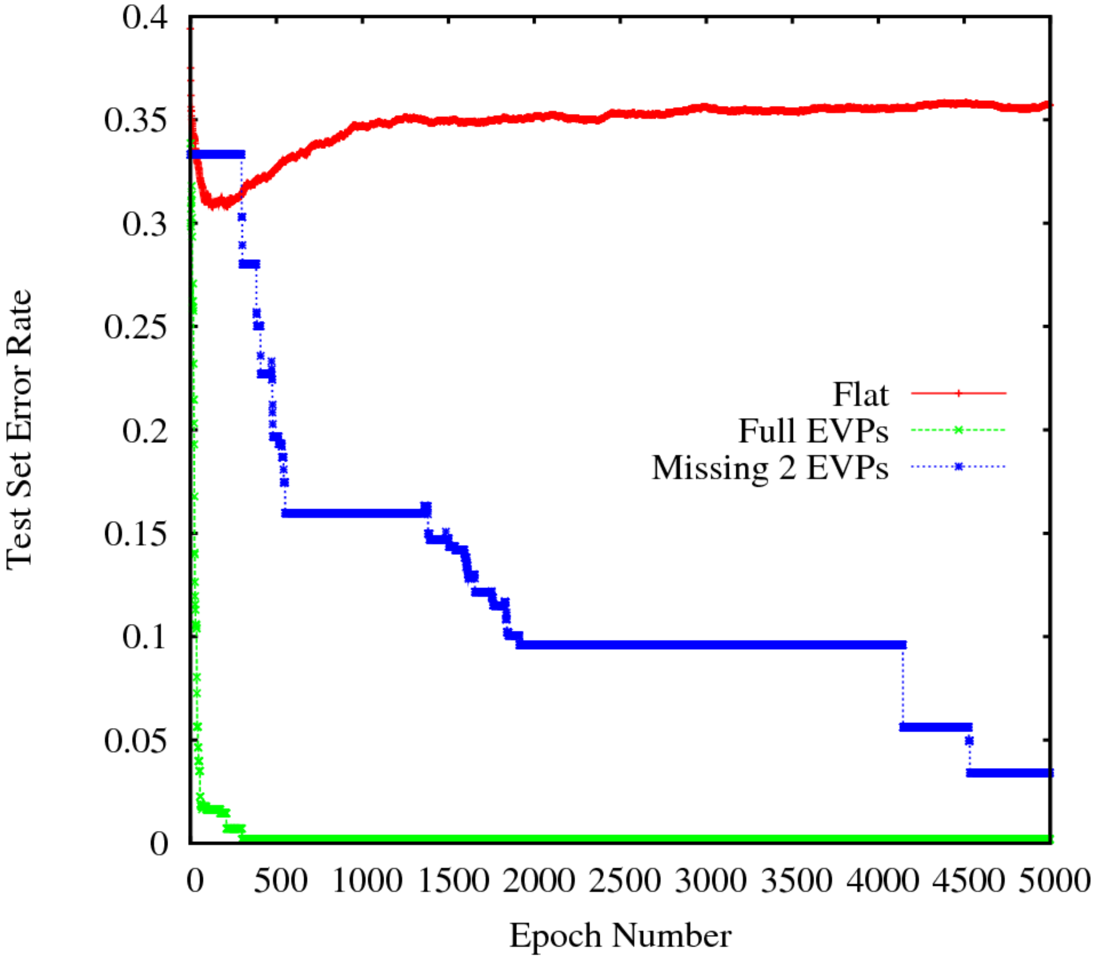


Figure 28: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 250 examples, where the hybrid learner is missing two EVPs.

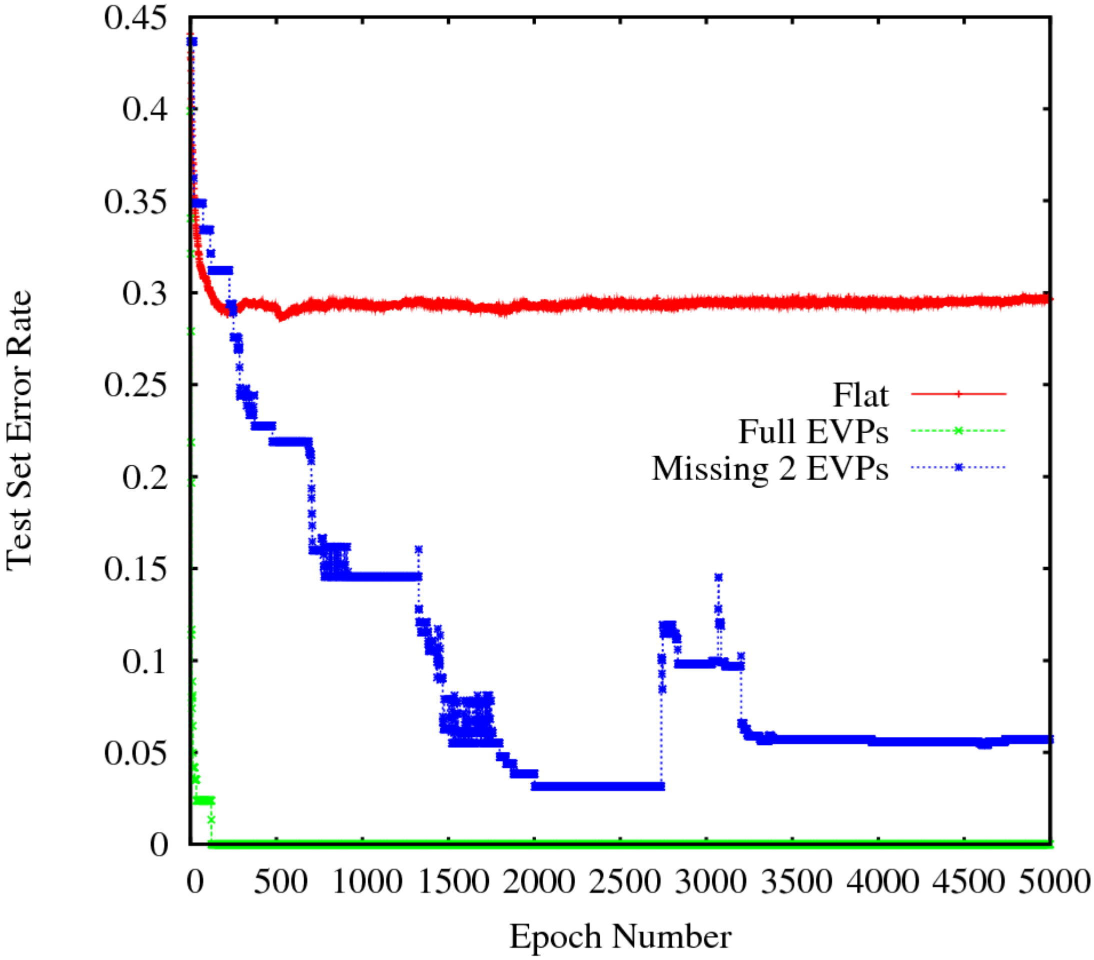


Figure 29: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 1000 examples, where the hybrid learner is missing two EVPs.

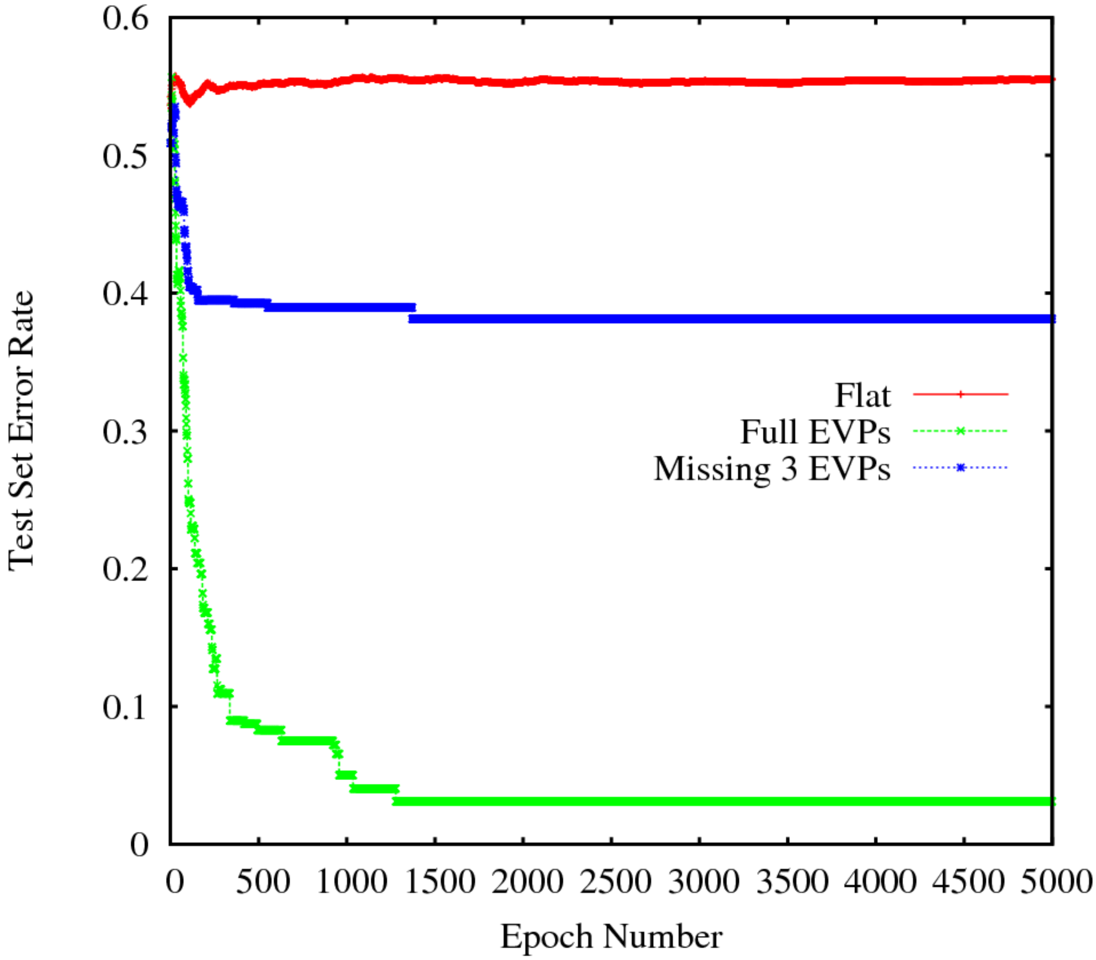


Figure 30: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 60 examples, where the hybrid learner is missing three EVPs.

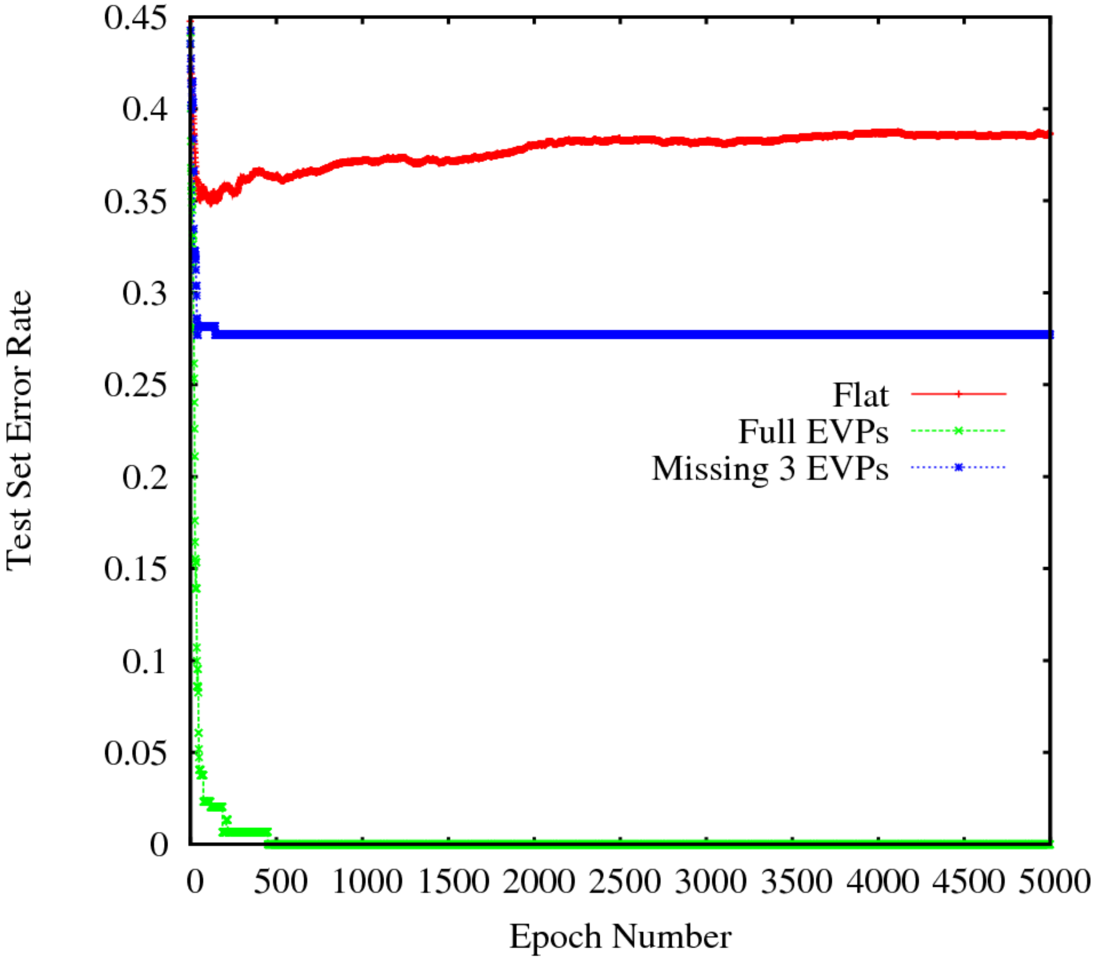


Figure 31: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 250 examples, where the hybrid learner is missing three EVPs.

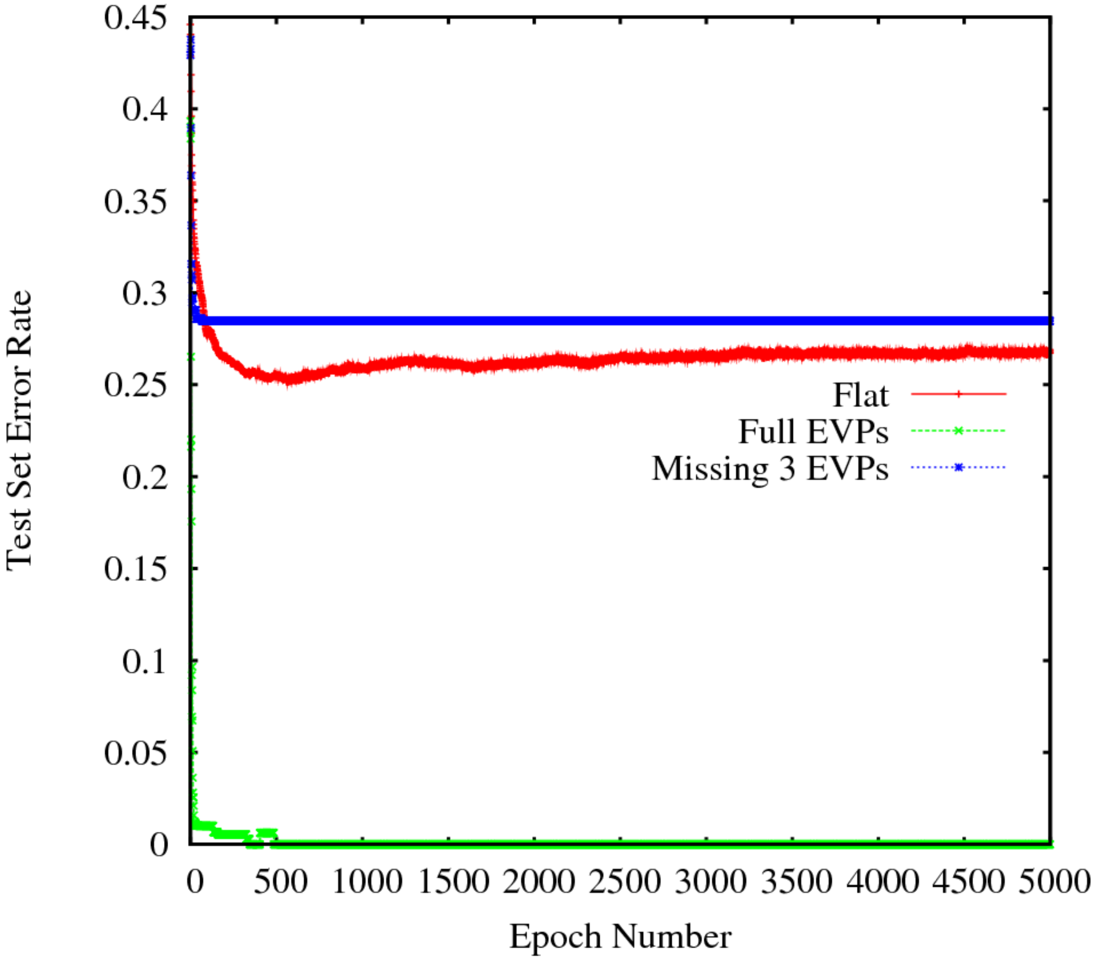


Figure 32: Training epoch vs. error rate for flat, AN and hybrid-AN learners with a training set size of 1000 examples, where the hybrid learner is missing three EVPs.

This is likely due to the susceptibility of large EVP-less regions (which are effectively deep ANNs) to local minima. There are techniques that specifically address this problem, such as KBANN [118], which initializes a deep network based upon prior knowledge, or the work of LeCun et al. [66] in which components in a structure are pretrained before end-to-end training is applied. However, in both cases the semantics of intermediate portions of the knowledge structure are enforced *before* end-to-end training, even if this enforcement is relaxed after initialization/pretraining. Thus, even though the requirement for enforced semantics at internal nodes is weaker in these other techniques than when using a AN structure with complete EVP coverage, there is still some reliance upon internal semantics to produce an initialization of the structure to be trained. The benefit of such techniques is that the problem of local minima in deep structure can be ameliorated to some extent by the initialization. ANs make even stronger use of these explicit internal node semantics, and, as evidenced by the very low final errors achieved by “complete” ANs in each of the experiments in this section, show a strong resistance to local minima. If the degradation in performance to a level worse than that of the flat learner in the experiment of Figure 32 is due to a problem with local minima for the hybrid learner, it is possible that after a very large number of repetitions, the average-case performance of the hybrid learner *may* exceed that of the flat learner.

An interesting secondary point to be noted is that, in each case, more epochs are required for the ANs with EVPs removed to reach their lowest error rate. This computational savings is another benefit of using EVPs wherever possible within a hierarchical classification learner, and is further evidence that the semantic pinning of nodes via EVPs is providing a restriction bias beyond that offered by the hierarchical structuring of knowledge alone.

Examining Figure 33, it is clear that the degradation in final error rate is much more pronounced in the AN that has had three EVPs ablated. It is likely that this is because a *connected subtree* of three nodes had their EVPs removed, creating a more substantial region within the AN that lacked semantic pinning. A large, connected region that lacks EVPs is likely to be very susceptible to local minima. In addition, these deep networks have more inputs than each member of the set of AN nodes they are replacing. Because the input dimension is higher, these deep networks will require more training examples to learn the target concept than the AN nodes they replace. It is also possible, though it seems unlikely, that the specific placement of nodes with removed EVPs may have an impact upon performance. For instance, perhaps removing the EVP from a single node in the layer immediately beneath the root node in these 4-layer binary ANs would have a more or less substantial impact than removing the EVP from a node immediately above the leaves. The experiments reported here do not speak to this issue. However, it does appear that removing EVPs from progressively larger connected subtrees within the AN will have the most catastrophic impact on final error rate.

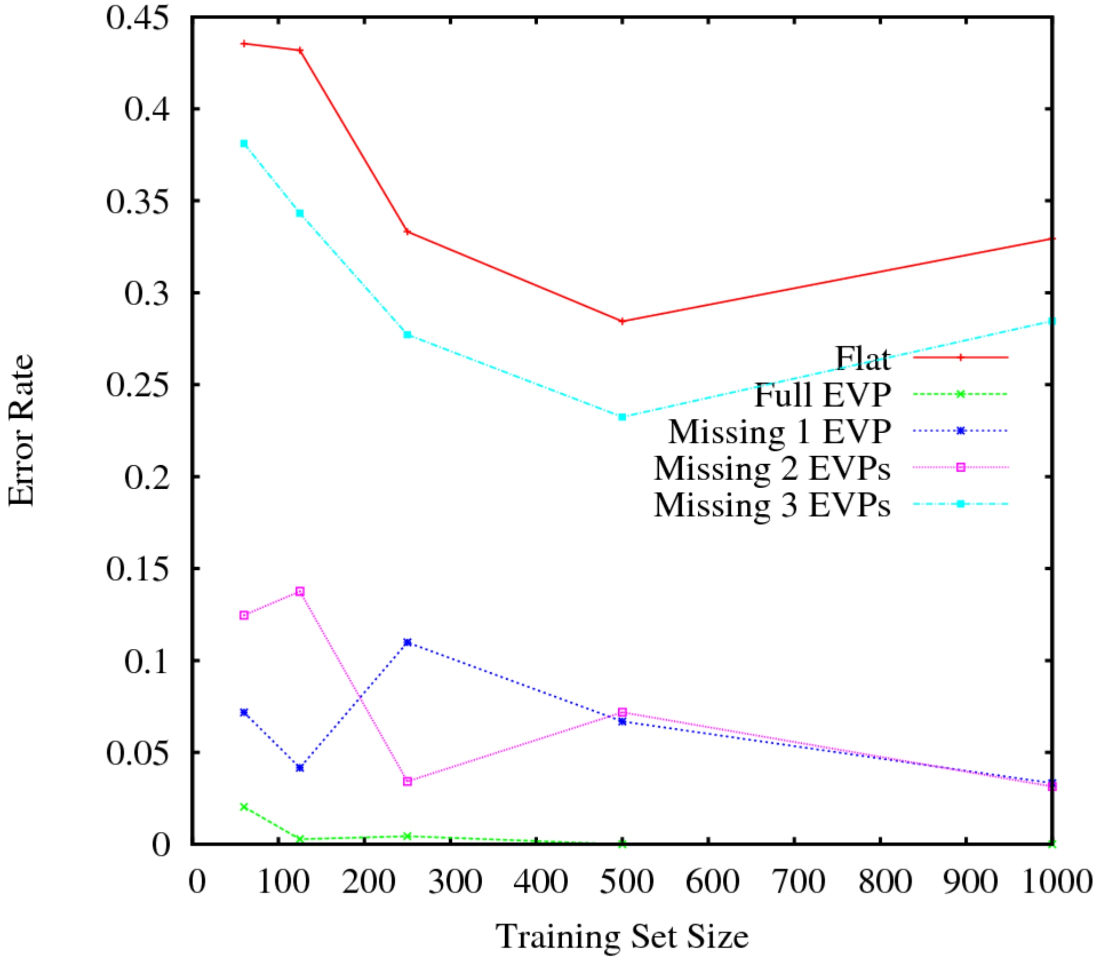


Figure 33: Training set size vs. best achieved error rate for flat, AN and hybrid-AN learners.

CHAPTER VI

AUTOMATIC CONCEPT REFINEMENT

Most of the experimental results described in the preceding chapter have been focused on verifying the usefulness of metaknowledge in the form of EVPs for diagnosing and adapting knowledge used for classification when the knowledge leads to errors. In this chapter, we focus on another major benefit of the explicit representation of the semantics of an agent's concepts through EVPs: the capability to automatically adjust those semantics when it is deemed advantageous to do so. Of course, this shifts the bias provided by an AN structure from a restriction bias to a preference bias. But, because the learning method employed always first tries to correct faulty classifications by adjusting knowledge stored within nodes to match existing concept definitions (as expressed by associated EVPs), we still realize the benefits of knowledge structure in increasing per-example generalization. The benefit of including the concept refinement procedure described in this section is that the AN representation becomes less sensitive to a particular kind of error in knowledge engineering, a misspecified quantizing EVP. While in principle automatic EVP adjustment should be possible for many kinds of EVPs, the method implemented in the Augur system, with which all of the experiments of this section are performed, works only with quantizing EVPs. The next section describes a procedure implemented within Augur for automatic adjustment of the constitution of equivalence classes represented within the AN structures used in this dissertation. Here, changing the constitution of an equivalence class means altering the set of circumstances under which an EVP associated with a node will return a particular value. This changes the concept being learned at the node such that situations encountered by the classifier will be equated differently with respect to the concept in question. This is particularly important when rote learners are employed in the nodes, as they are most sensitive to the quantization of their inputs. The subsequent section describes experimental results obtained by using this procedure in the synthetic domain.

6.1 Concept Refinement Mechanism

The task here is to detect that the agent is failing to learn given the fixed AN knowledge structure in use, and then automatically adjust an appropriate EVP within that structure to modify the semantics of the associated concept such that the failure is corrected. Given this task, the first requirement is that the agent have some way to monitor itself and detect learning failures that may require adjustment of one of its concept definitions. In this work, we require the agent to monitor the examples that are passed to each node in the network. If a sufficient level of inconsistency is detected in the examples passed to some node, we decrease the information loss at one of the child nodes. We use the following measure for inconsistency Φ at a particular node, for a particular input assignment I:

$$\Phi\left(I\right) = \frac{f\left(I, O^{*}\left(I\right)\right)}{\sum_{O} f\left(I, O\right)} \tag{1}$$

Where $O^{*}\left(I\right)$ is the most common output value for input assignment I seen in all examples passed to the node, and $f\left(I, O\right)$ is the number of times that example $\left(I, O\right)$ has been passed to the node. Thus, the denominator in Equation 1 is counting the total number of examples for input assignment I that have been passed to the node, and overall we are measuring the ratio, for a given input assignment at a given node in the AN, of the frequency with which the most common output assignment is seen relative to all the examples provided for that input assignment. Each time a new example $\left(I, O\right)$ is passed to a node, we update $\Phi\left(I\right)$ for that node, and determine whether the value has fallen below a threshold value. If there were no noise in the input examples, the threshold value could be set to 1, as any inconsistent input would be a sign of problems in the AN representation. However, in noisy domains it may be preferable to choose inconsistency thresholds < 1 to avoid spurious concept refinement due to noise.

The decision to refine a child concept based upon this inconsistency metric at a node is based upon the observation that learning failure manifesting as the receipt of inconsistent examples at an AN node (if not due to noise) is due to insufficiently discriminatory information coming into that node. That is, there are two situations between which a node is being asked to discriminate, but for which it is being given identical information – clearly an impossible task. This could be due to incorrect network structure, a problem for which this dissertation does not propose a solution, or due to a child node not passing along enough information about the world state. Given this state of affairs, when a node's inconsistency measure drops below the threshold, the agent will evaluate the following measure ϕ of average inconsistency along a particular slice of the input space for each child $c \in N$ (N is the set of children of a particular AN node, see Definition 3) of the node in question, in order to determine which one should have its concept semantics adjusted:

$$\phi_c\left(I\right) = \frac{\sum\limits_{i_1} \cdots \sum\limits_{i_{c-1}} \sum\limits_{i_{c+1}} \cdots \sum\limits_{i_n} \Phi\left(\left\langle i_1, \ldots, i_{c-1}, I\left[c\right], i_{c+1} \ldots, i_n \right\rangle\right)}{\sum\limits_{i_1} \cdots \sum\limits_{i_{c-1}} \sum\limits_{i_{c+1}} \cdots \sum\limits_{i_n} 1} \tag{2}$$

Where $I\left[c\right]$ is the value coming from child c in input assignment I. This measure computes the average inconsistency value over all input assignments for which child c provides value $I\left[c\right]$. After computing this value for each $c \in N$, we can determine:

$$\arg\max_{c \in N} \phi_c\left(I\right) \tag{3}$$

We then choose a random child from the set represented by Expression 3 as the candidate for concept refinement. As this procedure makes an educated guess as to which children are the most likely to require concept refinement (and then a pure guess as to which of those to refine), there is no guarantee that the proper child will be chosen in every case. However, if the wrong child of a failing node is chosen, failure will occur again and we will have a chance to increase the output range at another node. This procedure may lead to some spurious increases in output value ranges, which will have a negative impact on generalization and learning speed. However, the goal of concept refinement in this scenario is to prevent the exclusion of correct hypotheses while allowing the use of a strong restriction bias. The problem of choosing the wrong child to split could potentially be alleviated by adjusting the threshold value during the course of training. Early in training, the value could be set to zero, preventing any concept refinement. After some number of examples have been seen, it is increasingly likely that estimates of the relative inconsistency along various slices of the input space (ϕ_c) will be accurate, and the threshold could be raised to allow some concept refinement to occur.

Table 4 summarizes the procedure described in this section. The procedure *AN-inconsistency-check* described in Table 4 is invoked every time the learner's update function ($L.U$, from Definition 2) is called. That is, the inclusion of this concept refinement procedure adds a call to the *AN-learning* procedure of Table 2 as shown in Table 5. The next section says more about how the subroutine *increase_quantization* of Table 4 is implemented in Augur, and describes the experiments run in the synthetic domain.

6.1.1 Empirical Evaluation of Concept Refinement

All of the EVPs dealt with in the experiments described here are pure *quantizing EVPs*, as described in Section 2.1, that perform a single branch based upon an observed value and then emit a category. For this type of EVP, modification in these

Table 4: Top-level reasoning used to maintain the inconsistency measure at the root of an AN hierarchy when the root is being trained on an example and to select and adjust a child node if necessary.

```
/* Values from Definition 3:
 *   a.N            - a set of ANs.  The children of ‘a’.
 *   a.P            - the EVP for ‘a’.
 *   a.L            - the learner associated with ‘a’.
 *
 * Added notation:
 *   a.$\Phi(I)$          - the value of $\Phi(I)$ as in Equation 1
 *                  at the root node of ‘a’.
 *   a.$I[O]$          - the number of times that output $O$
 *                  has been seen in a training example at the
 *                  root of $a$ for input $I$.
 *   $Threshold$      - a global, user-defined threshold
 *                  defining the level of inconsistency tolerated.
 *                  $0 < Threshold \leq 1$.
 *
 * Subfunctions used:
 *   increase_quantization(EVP $P$, Learner $L$, Value $V$):
 *                  adapts the EVP provided as the first
 *                  argument such that situations that would
 *                  previously have been grouped into the
 *                  equivalence class designated by the third
 *                  argument are now grouped into two distinct
 *                  equivalence classes, and adjusts the supervised
 *                  classification learner provided by the second
 *                  argument to be prepared to handle the newly
 *                  introduced equivalence class as a potential
 *                  input value.
*/

begin AN-inconsistancy-check(Abstraction Network $a$, Example $(I, O)$)
      $a.I[O] \leftarrow a.I[O] + 1$
      if $a.\Phi(I) > Threshold$
               $n \leftarrow \arg\max_{c \in a.N} \phi_c(I)$
               increase_quantization($n.P, a.L, I$)
end
```

Table 5: Self-diagnosis and self-repair procedure used to correct knowledge stored in an Abstraction Network a, modified to include the call to the concept refinement code of Table 4.

```
/* Values from Definition 3:
 * a.P              - the EVP for ‘a’.
 * a.last_value     - the last value produced by ‘a’.
 * a.N              - a set of ANs.  The children of ‘a’.
 * a.L              - the learner associated with ‘a’.
 * a.last_input     - the last input sequence provided to ‘a’.
 *
 * Values from Definition 2:
 * L.U              - the learner’s update (learning) function.
*/

begin AN-learning(Abstraction Network $a$)
      Bool $flag \leftarrow true$
      if $a.P() = a.last\_value$, return $true$
      forall $n \in a.N$
               if AN-learning($n$) $= false$, $flag \leftarrow false$
      if $!flag$, return $false$
      $a.L \leftarrow a.L.U((a.last\_input, a.P()))$
      AN-inconsistency-check($a, (a.last\_input, a.P())$)
      return $false$.
end
```

circumstances involves splitting one of the branches into two, essentially creating a new “bin” for quantization. Specifically, we split the branch associated with the output value that indexed the learning failure location at the parent, $I[c]$. The procedure used in these experiments splits the branch in question by continuing to use the existing branch for half of the values that would previously have caused it to be taken, and using a newly introduced branch for the other half of the values. The newly introduced branch, of course, results in the emission of a new category, not present on any other branch in the EVP. For this reason, a learner-type-dependent operation may be required to prepare the learner at the node with an adjusted EVP to receive examples with an input entry along the adjusted dimension equal to the new value. In the case of table-based rote learners, which were used in these experiments, this means adding a new (multi-dimensional) row of entries along the dimension corresponding to the adjusted child that will learn the appropriate outputs for inputs containing the newly added equivalence class. These steps are the procedure implemented by the subroutine *increase_quantization* of Table 4. If the middle of the current range is not the best point at which to split the branch, we rely on further splits to eventually choose the right decision point. In fact, when branching on real-valued observations, it is unlikely that the *correct* decision point will ever be found. For this reason, working with an inconsistency threshold of 1 is inadvisable even in a noiseless domain. Choosing some value less than 1 will allow for some stray examples that may be due to a slightly misplaced decision point in a child’s EVP. As with the possibility of selecting the wrong child, choosing the wrong decision point between the branch undergoing split and the newly introduced branch will decrease learning speed, but will not result in the exclusion of the correct hypothesis. After adjusting the EVP at the child node to refine the set of equivalence classes of which its concept consists, continuing learning will cause the node to begin to emit the newly added branch value, providing finer-grained information to the parent node. Through this procedure, we have automatically adjusted the concept represented at a node in the EVP based upon a failure of that concept to adequately address its functional role (supporting a correct prediction at the parent node) within the AN.

We have evaluated the bin splitting procedure described above in the synthetic domain of Section 4.1. In these experiments, table-based rote learners were used within all nodes. In these experiments, after constructing the fixed and learner ANs, we added an additional branch, and thus new concept category, to one or more random nodes within the fixed AN. Based on the EVP construction within the learner AN, this change will result in two categories produced by the changed node within the fixed AN being perceived as within a single category by the learner AN. Thus, to perfectly match the overall target concept, the learner AN needs to refine one or more of its concepts[1]. Figures 34 and 35 depict the results of learning with ANs that do and do not incorporate the concept refinement procedure when one and two EVPs in the fixed AN have undergone branch splitting, respectively. Results shown are from an average of 20 randomized trials. In these experiments, the inconsistency threshold value was set to 0.3.

These results clearly show the effectiveness of the concept refinement procedure. When it is not enabled, the learner AN fails to reach zero final error, as the correct hypothesis cannot be represented. When it is enabled, concept refinement is correctly triggered, allowing the learner to find the correct (zero error) hypothesis. In this clean synthetic domain, the learner is observed to make only and precisely the required concept adjustment in each case, as apparently evidenced by the smooth tracking of the non-concept refining learner's error rate by the concept refining learner until the non-refining learner approaches its limit. In real world domains, there is likely to be more

[1] There is a degenerate case where the random initialization of the fixed AN happens by chance to make no discriminations between situations captured by the split category. Performing a number of re-randomized repetitions reduces the impact, if any, of these degenerate cases.

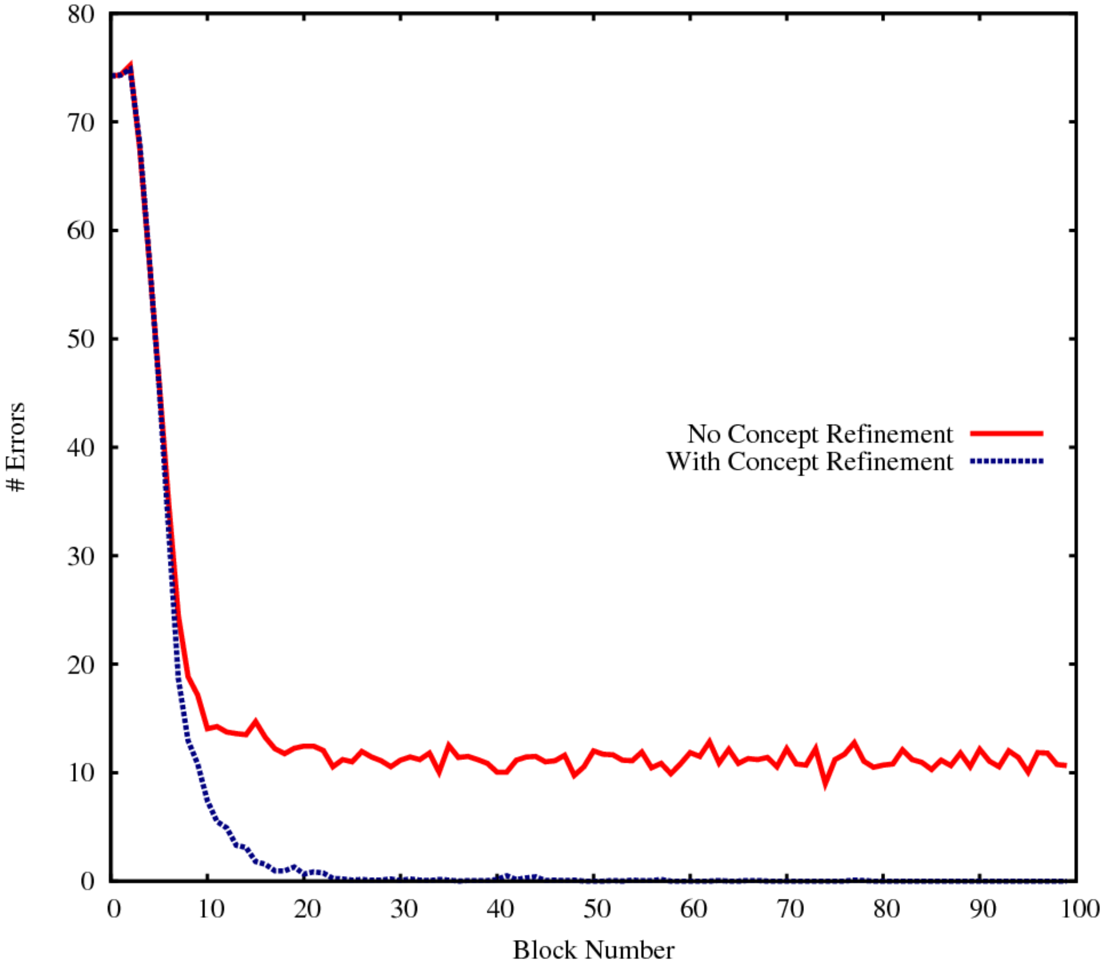


Figure 34: Error rate vs. block number with one overly coarse concept.

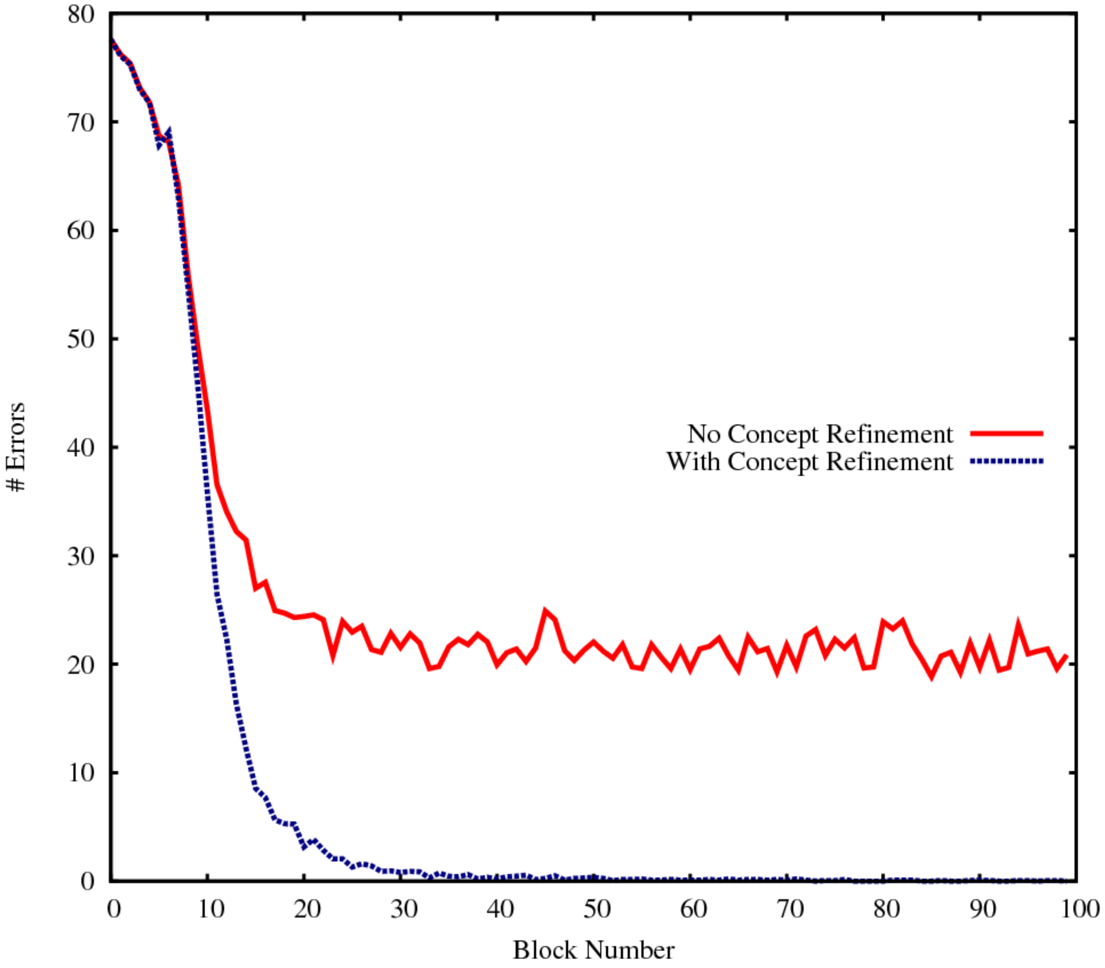


Figure 35: Error rate vs. block number with two overly coarse concepts.

negative impact on the learning rate of an agent that makes use of the concept refinement procedure, as in general there will be spurious refinements triggered. However, the final error rate of a concept refining learner is still expected to be less than or equal to that of a non-refining learner, with learning rate still substantially better than that of an unstructured learner.

CHAPTER VII

ANALYSIS OF HIERARCHICAL CLASSIFICATION

In this chapter we describe results with respect to optimality of the non-exhaustive diagnostic procedure described in Section 2.4, notes on the convergence properties of AN learning, and some work on counting the number of hypotheses expressible in an AN classification hierarchy through the use of partition lattices. These results speak to the particulars of the problem setting and knowledge structures in the context of which we chose to test our theories, rather than directly to the core idea of this work, the use of EVP metaknowledge for reflection and self-repair. However, we include them here as they represent a secondary contribution arising from the research described in this document.

7.1 Optimality of Non-Exhaustive Diagnostic Procedure

The structural credit assignment technique described in Section 2.4 is optimal with respect to maximizing expected decrease in *diagnostic search space* entropy with each probe, under assumptions outlined below. Here, the diagnostic search space consists of all possible error conditions that lead to an error observed at the root of the AN, under the following provisions:

- A value produced at a node is wrong only if it is *objectively* wrong (wrong according to the associated EVP) and either is the root of the hierarchy, in which objective incorrectness is sufficient, or is also *subjectively* wrong (recursively, leads to the production of an erroneous value at the parent, and thus eventually the root). In the language of diagnosis, this amounts to ignoring compensating faults. This view of error arises from our functional stance towards knowledge;

since knowledge in an AN exists to serve a purpose, it is incorrect only if it fails to serve that purpose.

- Without loss of generality, we assume that each node produces values in a way that is actually dependent on the values of all of its child nodes.
- The learner types used within nodes are such that they are amenable to the use of a non-exhaustive diagnostic procedure. It is not clear how to precisely define the characteristics that are required, though it is known that some learner types (ANNs) do not behave well under the non-exhaustive diagnostic procedure, while others (rote learners and kNNs) do not have problems. A potential issue is that many learner types require that training examples be drawn from a stationary distribution in order to guarantee convergence. A non-exhaustive diagnostic procedure *does not* in general provide a stationary distribution of training examples. The distribution of examples at a particular node in a hierarchy being trained according to such a procedure is dependent upon the correctness of knowledge at nodes around it in the hierarchy, and thus as training occurs, the distribution may change.

As an aside, note that the final assumption listed above encapsulates an important point about the interaction between the "causal backtracing" style of diagnosis commonly used in knowledge-based AI (here applied to self-diagnosis) [105] and statistical machine learning techniques, where the requirements of underlying implementation choices can preclude the application of seemingly intelligent optimizations at a higher level. In any case, some learner types (such as our table-based rote learners) do not require that examples be drawn from a consistent distribution. Thus, the diagnostic procedure that one wishes to use becomes a design consideration that the implementer of an AN must bear in mind when selecting the learner types to be used within nodes. If the cost of EVP execution is particularly high within a domain, it may be worthwhile to use learners such as rote learners that are not sensitive to example distribution, allowing the use of a parsimonious, non-exhaustive diagnostic procedure. The precise characterization of the requisite characteristics of learners that will work with the non-exhaustive diagnostic procedure of Table 2 is left as future work.

Because of the first assumption listed above, the diagnostic search space for a given failure consists of all possible connected subtrees that include the root. A given element in the diagnostic space accurately identifies the cause of error for a given failure situation if it is the maximal such set for which all nodes in the set produced incorrect values during the related inference. Here, a failure situation is any situation in which the diagnostic procedure is invoked due to observed failure – in this work, this occurs whenever an error is noted at the root of the classification hierarchy. The set represents the nodes that produced erroneous values during the failure, and learning within nodes will occur at the lower fringe of the set, as per the credit assignment and learning procedure described earlier. Note that this diagnostic search space is distinct from the *hypothesis* space searched by the EVP-based self-diagnosis and learning procedure applied to a sequence of examples; *this* search space is not concerned with any specific knowledge stored at the nodes, but only with explaining the fault location(s) that led to a particular failure instance.

The task for our diagnostic procedure is to identify the hypothesis within the diagnostic search space that accurately explains the error being diagnosed. This will be achieved by executing the EVPs associated with some of the nodes within the AN, effectively probing those nodes. Because we anticipate that there will be some cost associated with EVP execution (which throughout this work we assume to be uniform), we would like to select the single correct diagnosis using as few EVP executions as possible. To achieve this, we will at each step select an EVP execution that reduces the number of viable hypotheses remaining in the search space by the maximum amount possible. Because we wish to select *a single correct* diagnosis, and not achieve the weaker condition of obtaining some degree of belief about the diagnosis, prior probabilistic beliefs about the relative likelihoods of the diagnoses are not significant in guiding probes each of which can only rule out (and not directly confirm) some diagnoses – to leave only one viable diagnosis, we must rule out *all* other diagnoses, no matter how improbable we may consider them a priori. For this reason we can, without loss of generality, treat the diagnoses within the space as being equiprobable for the purpose of our entropy-based analysis in this section. This analysis could alternatively be done without using the notation of entropy, by instead referring to the cardinality of the set of viable hypotheses after each probe. This quantity, in fact, is what we are measuring with the diagnostic search space entropy given our use of a uniform prior over diagnoses.

Let us define D to be a random variable that indicates which of the diagnoses within the diagnostic hypothesis space is correct for a given failure instance. We will also define a random variable N_i for each node n_i in the AN that is the subject of diagnosis. Each N_i indicates whether the value produced by the associated node was correct for the failure instance being diagnosed. Finally, define $H(D)$ to be the entropy of the diagnostic search space. Given these definitions, we can write the expected entropy remaining in the diagnostic search space after probing some node n_i as $H(D|N_i) \cdot P(N_i) + H(D|\neg N_i) \cdot P(\neg N_i)$. (As described above, we treat the diagnoses as having a uniform prior for the purpose of this analysis). If we probe n_i and it is found to have produced a good value, the remaining entropy is written $H(D|N_i)$, or if it is found to have produced a bad value, $H(D|\neg N_i)$. We will use $H^{+}_{n_i}$ as a shorthand notation for the former quantity and $H^{-}_{n_i}$ for the latter, as the diagnostic search space in question is unambiguous.

Lemma 1 *For any pair of unprobed nodes n_i, n_j in an AN such that n_i is a (possibly indirect) ancestor of n_j:*

1. $H^{+}_{n_j} \geq H^{+}_{n_i}$

2. $H^{-}_{n_j} \geq H^{+}_{n_i}$

3. $2 \cdot H^{+}_{n_i} + H^{-}_{n_i} \leq H^{+}_{n_j} + H^{-}_{n_j}$

To see that (1) & (2) are correct, notice that if n_j is probed and found to have produced a correct value, all hypotheses consistent with this observation are consistent with *either* result of probing n_i; that is, the set of consistent hypotheses remaining viable if n_j is probed and found to have produced a correct value is a superset of the hypotheses consistent with n_i having been found to be correct. Likewise for n_j being found incorrect. This is because, given the way the hypothesis space is defined, if n_i produced a correct value we do not care whether n_j produced a correct value. So, if we probe at n_i first, we may not need to probe n_j, whereas if we probe n_j first, we will still need to probe n_i, for any outcome at n_j. Probing at n_j will only allow us to rule out a subset of the hypotheses consistent with n_i being incorrect, and none of those consistent with n_i being correct. The intuition behind this proof lies here. Since we do not know whether node n_j's status is important until we probe node n_i, it is more fruitful to probe at n_i first. This is true regardless of whether we believe that n_j is more likely to be incorrect than n_i, because even if n_j is found to be incorrect, the system must then probe at n_i anyway, to see whether the result at n_j is even significant. (3) is a related inequality, and is based on the observation that if n_j is a direct descendant of n_i, the hypotheses consistent with n_j being correct are those hypotheses consistent with n_i being correct, plus some fraction of those consistent with n_i being incorrect. Likewise for n_j being incorrect. Since there is no diagnosis consistent with neither n_j being correct nor n_j being incorrect, and there is no diagnosis consistent with n_i being incorrect, n_j being correct *and* n_j being incorrect, we arrive at a variant of (3) with an equality rather than an inequality for cases where n_j is a direct descendant of n_i. Now notice that if n_j is an indirect descendant of n_i, the sets of consistent hypotheses contributing to $H^+_{n_j}$ and $H^-_{n_j}$ will not only be supersets of those contributing to $H^+_{n_i}$, but will also include some of those contributing to $H^-_{n_i}$ that are consistent with correct values having been produced by some nodes on the path from n_j to n_i. This leaves us with (3) as presented above, since the RHS may exceed the LHS due to this duplication.

Theorem 1 *For any two unprobed nodes n_i and n_j within an AN, where n_i is a (potentially indirect) ancestor of n_j, the expected remaining entropy in the diagnostic search space is less at n_i, making n_i a more desirable probe point.*

To prove Theorem 1, let us assume that n_j constitutes a better next probe point than n_i, on the basis of expected remaining entropy in the diagnostic search space. Then $H^+_{n_j} \cdot P(N_j) + H^-_{n_j} \cdot P(\neg N_j) < H^+_{n_i} \cdot P(N_i) + H^-_{n_i} \cdot P(\neg N_i)$. Substituting using our lemma and simplifying yields $P(N_i) > 1$, a contradiction.

Moving from this result to our non-exhaustive self-diagnosis procedure is straightforward; it is preferable to probe nodes with ancestors that have been probed (and found incorrect, of course). Further, the order of probing among such nodes is arbitrary since the probing of any node with this characteristic cannot remove the characteristic from any other node, and no such nodes can remain when a diagnosis is uniquely selected.

7.2 Convergence of AN Learning

Past work on hierarchical classification, in particular tree-structured bias (TSB) [97][115] and structured matching [37], makes it clear that hierarchical knowledge structures can effectively represent classification knowledge if structurally correct. Here we typically assume that the *structure* of the knowledge representation has been correctly engineered by a human designer, except in the experiments that specifically test learning degradation when knowledge engineering is flawed (Section 4.3). Even in these scenarios, empirical evidence supports the benefits of using structured knowledge representations for learning. Tadepalli and Russell [115] prove that learning over classification hierarchies is tractable. In this section, we provide some notes on the convergence properties of ANs under various assumptions.

7.2.1 Full EVP Evaluation vs. Causal Backtracing

To understand convergence characteristics under the two diagnostic techniques used in this work, full EVP evaluation and the causal backtracing style (Section 2.4), it is useful to consider the sequence of examples induced at an arbitrary (non-leaf) node within the AN learner.

When full EVP evaluation is used, every EVP within the AN is executed for every learning example. Examples are then constructed at each node by using the results of child node EVP execution to form the correct input feature vector, and the result of local EVP execution to form the correct output class for the example. In such a case, assuming that the environmental dynamics are stable and that EVPs are fixed (we deal with mutable EVPs in Subsection 7.2.2), the sequence of examples induced at each node within the AN is drawn from a stable distribution – the fixed EVPs are sampling from a stable distribution provided by the environment. Under full EVP execution diagnosis, there is no difference in the sequence of examples seen by a supervised learner assigned to a particular classification subproblem within an AN and the sequence of examples that would be seen by a standalone learner working on the same problem in isolation. Thus, if the characteristics of the environment are assumed to be such that the learners used within AN nodes will converge after some number of examples, we can be equally confident that all learners within the AN will have converged once the requirements of the neediest learner have been met. If the learners used within AN nodes are sufficiently expressive and the AN structure has been engineered correctly, we know that the AN as a whole can represent the correct hypothesis for the overarching classification problem, as discussed at the beginning of this section. Based on these observations, it seems reasonable to expect that ANs will have reasonable, tractable convergence properties under these conditions, though we do not develop explicit PAC bounds for an AN classifier as a whole.

When the causal backtracing style of diagnosis described in detail in Section 2.4 is used instead, convergence can no longer be assured for a subclass of supervised learner types. This is true because (1) nodes are only provided examples for which they have been found to make an error, and (2) the effective distribution from which examples are drawn is not stable. Of course, point (2) is related to point (1). As nodes within an AN begin to successfully classify some examples, the distribution of examples seen by those nodes and neighboring nodes will shift. When learners (such as ANNs) sensitive to the distribution of training examples are used within an AN, this characteristic of causal backtracing diagnosis can lead to oscillatory non-convergence. Of course, there are learner types (such as rote learners) that are not sensitive to input example distribution. Other factors, such as substantial noise in the results of EVP execution, may still cause some problems within a network that uses pure causal backtracing diagnosis. However, when appropriate learner types are used, empirical results suggest that good results can be achieved with causal backtracing diagnosis. For this reason, it is our recommendation to try learning with causal backtracing if EVP execution cost is high, using full EVP execution as a fallback. Further, if EVP execution cost is both high and nonuniform, one would likely wish to go beyond basic causal backtracing, also considering a value of information analysis. However, in this thesis we do not deal with non-uniform EVP costs though we do acknowledge the potential importance of the issue.

7.2.2 Mutable EVPs

The preceding subsection considered only cases in which EVPs within the AN learner are fixed for the duration of learning. Here, we address the situation that arises when the method of Chapter 6 (or another method for automatic EVP adjustment) is used to allow EVPs to be altered during learning. The only method for EVP adjustment that we have implemented involves making quantization changes. However, here we are considering a broader set of possible EVP modifications. We do explicitly intend that these modifications result in an alteration of the semantics of the associated node, as these semantics are fixed by the node's EVP. When EVPs may be modified by the learner, the classification subproblems assigned to nodes within the AN may change during learning. Changes arising from the method of Chapter 6 are incremental rather than radical, and thus much of the knowledge already acquired at the affected node is likely to remain valid. However, this need not be true in general. Because of the substantial setback in learning that can occur when subproblems are altered, particularly at a late stage of learning, we recommend that the conditions for initiation of automatic EVP adjustment be fairly stringent, i.e. EVPs are changed only when there is substantial evidence that the concept represented is inadequate. As long as it can be guaranteed that automatic EVP refinement will cease at some point in the learning process, the qualitative comments from the preceding subsection will still hold.

7.3 Restriction Bias from Hierarchical Representations

In this section, we provide some mathematical basis for understanding and beginning to quantify the restriction bias that is imposed upon a classification learner by a fixed hierarchical knowledge structure. This type of analysis has its roots in the VC dimension measure [123] of the expressivity of a hypothesis space. In VC dimension analysis, one determines the expressivity of a hypothesis space to be used by a learner in order to understand the capacity of a learner to overfit, as well as the generalization power afforded by the representation (these characteristics are inversely related). In this section, we provide an analytical direction by which one may understand the expressivity and thus generalization power of hypothesis spaces induced by hierarchical knowledge representations.

First, note that a classification (of discrete objects) is a particular assignment of labels to a partition of a set containing those objects (this set is also sometimes referred to as the instance space). For a classifier with no restriction bias that classifies n objects into x categories, we have x^n hypotheses that can be expressed. Letting $S(n, k)$ represent the $(n, k)^{\text{th}}$ Stirling number of the second kind, and $(x)_n$ the n^{th} falling factorial of x, the hypotheses can be counted by Expression 4 for $n, x \geq 0$[1]:

$$\sum_{k=0}^{n} S\left(n, k\right) \left(x\right)_k \tag{4}$$

A term in the sum of Expression 4 counts the number of ways that n items can be partitioned into k sets ($S(n, k)$), and multiplies this by the number of ways to uniquely assign x labels to the k sets ($(x)_k$). This computation can be visualized through a partition lattice. Figure 36 illustrates a lattice relevant when classifying four items.

In the case that our learner is assigning four items to two categories, Expression 4 expands (and then simplifies) to:

[1] Though it may seem slightly strange that this expression states that there is one way to classify zero items into zero categories, this result is in keeping with the assertion of Sterling numbers that there is one way to partition zero items into zero sets. In practice, this border condition is unlikely to be of import.

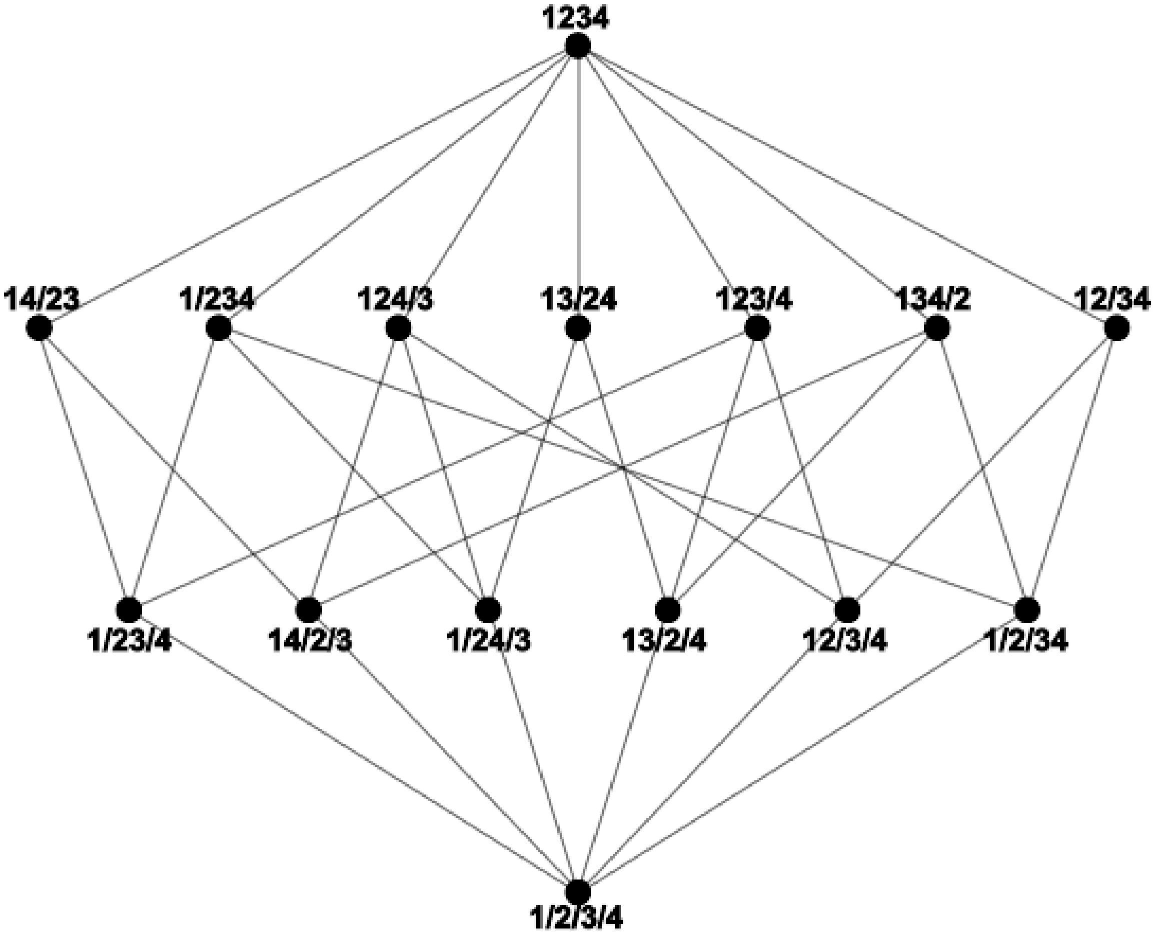


Figure 36: Partition lattice for four items.

$$\begin{aligned}S(4,0)(2)_0 + S(4,1)(2)_1 + S(4,2)(2)_2 + S(4,3)(2)_3 + S(4,4)(2)_4 &=\\ 0 \cdot 1 + 1 \cdot 2 + 7 \cdot 2 + 6 \cdot 0 + 1 \cdot 0 &=\\ 16&\end{aligned}$$

As expected, the result is 2^4, all possible hypotheses when four items are to be placed into two categories. Each nonzero Stirling number of the second kind in the expression above corresponds to a row in the partition lattice of Figure 36. For instance, the second row from the bottom in Figure 36 contains seven items as there are seven ways to partition four items into two sets. Because there are only two classes that can be emitted, we effectively disallow the selection of partitions from the two lowest rows in the partition lattice. It is this process that forces *information loss* of varying degrees within a classifier. If we were allowed to place four items into four distinct categories, no information loss would be forced by the classification – the classification process is reversible under some allowed classification functions. As we restrict our classifier to progressively coarser categorizations by limiting the number of output classes, we lose more and more information. The progressive process of information loss at nodes within a classification hierarchy is the basis of the imposed restriction bias. This method of counting the size of the hypothesis space in terms of Sterling numbers of the second kind and visualization in terms of partition lattices is overly complex when dealing with straightforward classifiers, but will become valuable as we now turn to the restriction bias imposed by hierarchical knowledge representations.

Let us now consider a classifier that makes use of a very simple structured knowledge representation, depicted in Figure 37.

This classifier has three inputs, each taking two possible values. The first and second stage sub-classifiers are each able to emit two class labels. Notice that the

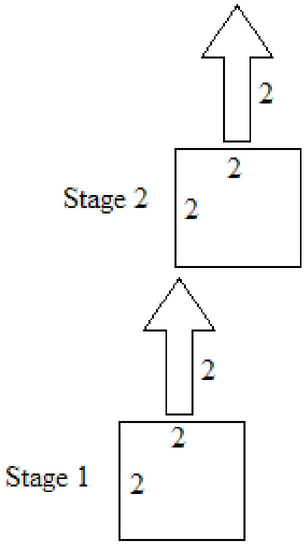


Figure 37: A simple classification hierarchy with 3 inputs, each taking 2 values. The first and second stage sub-classifiers are each able to emit 2 class labels.

labeling of partitions is only important in the final stage. The number of overall hypotheses expressible by the structured classifier of Figure 37 is *not* doubled because we can reverse the class labels on otherwise equivalent partitions coming from the stage 1 classifier. Only the final partitioning of the $2^3 = 8$ input values and the label assignments onto *those* partitions is significant in determining the total number of expressible hypotheses. The first stage partitions four values into at most two sets (partitions including three or four sets are excluded as they cannot be uniquely labeled at the stage 1 output). In considering the number of hypotheses expressible by the classifier at large, we need only concern ourselves with the most granular partitions that can be produced by a given sub-classifier. This is because further conflations can always be handled by (and will be counted at) the root classifier. At subordinate nodes, we are only concerned, in effect, with the partitions that are *excluded* by the node. Following this logic, the most granular partitions that can be produced by the first stage, as shown in the corresponding row of the lattice in Figure 36, are:

14/23, 1/234, 124/3, 13/24, 123/4, 134/2, 12/34

Now consider what happens when the partitions defined by the stage 1 classifier are combined with the additional raw input that feeds into the stage 2 classifier. In essence, each of the four original items classified at stage 1 multiplies into two items, as we form the Cartesian product of the stage 1 inputs with the additional input values. The key here is that the partitioning decisions made at stage 1 will apply (separately) to the new sets of items created when adding the new, raw input at stage 2. This is most easily illustrated by example. Let us refer to the four items created by pairing the first of the raw input's values with the four original items as 1,2,3,4; and the four items created by pairing the second of the raw input's values with the four original items as 5,6,7,8. Then, the most granular partitions that can be created by the second stage classifier are:

> 14/23/58/67, 1/234/5/678, 124/3/568/7, 13/24/57/68, 123/4/567/8, 134/2/578/6, 12/34/56/78

Further, the *only* partitions that can possibly be created by the second stage classifier are those listed above and further coarsenings of those listed above. Thus, with the addition of a binary input, we have moved from a partition lattice for four items into a partition lattice for eight items. Our possible starting points within this eight item partition lattice are the nodes corresponding to the partitions enumerated above. If we imagine the arcs within the partition lattice to be directed, in every case from more granular nodes towards less granular nodes, then the set of nodes reachable in the eight item partition lattice from these starting nodes is the number of partitions that can be produced by the stage 2 classifier. To count the number of hypotheses expressible by the classifier of Figure 37, we need to count the number of nodes reachable at each level l (numbering from the topmost, coarsest partition, and beginning with 1) and multiply by $(2)_l$. In Appendix A, we provide an enumerative solution to this small example problem, demonstrating that there are 88 hypotheses expressible by the classifier of Figure 37. This stands in contrast to the $2^8 = 256$ hypotheses expressible by an unstructured learner with three binary inputs and a binary output and with no alternative restriction bias. Thus, even the simple knowledge structure of Figure 37 has led to the exclusion of $256 - 88 = 168$ hypotheses, or a 65.6% reduction in hypothesis space size.

To quantify, or even provide bounds on the hypotheses expressible by a hierarchical classifier in general, this analysis will need to be substantially expanded and generalized. For instance, how do we generate the possible starting nodes within a higher level partition lattice when we combine not a sub-classifier and a raw input, but the outputs of two sub-classifiers? The problem rapidly becomes more complex. This generalization is left as future work, as hierarchical classification itself is not the focus taken by this dissertation. However, this line of reasoning may be useful in the eventual quantification of the restriction bias imposed by hierarchical classification knowledge structures.

CHAPTER VIII

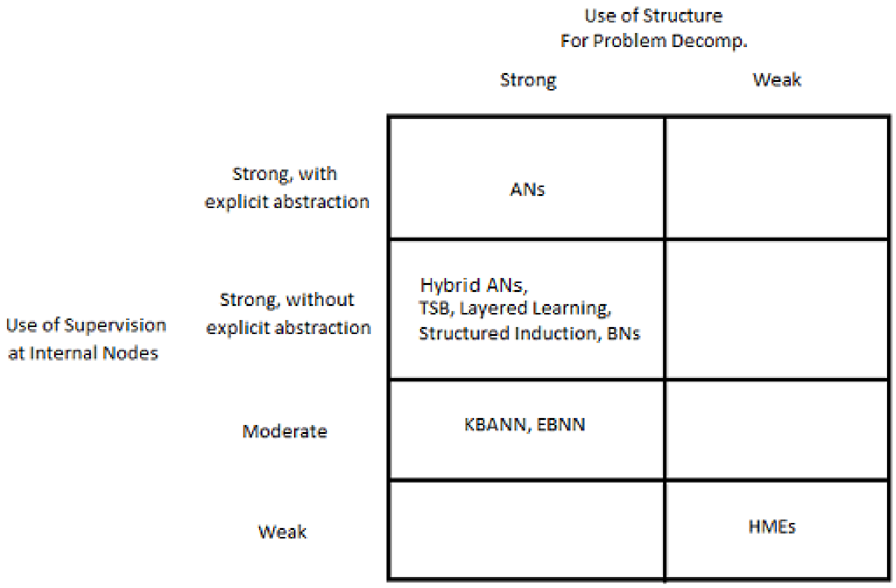
RELATED RESEARCH

Beyond the classifiers which we have integrated directly with the Augur system (kNNs [28] and ANNs [94]), there are several lines of research that are relevant to the AN representation and algorithms that we use to address the compositional classification problem chosen as a test domain for EVP theory. The following section compares and contrasts ANs with these other lines of research – though of course, the use of EVPs, and the power they bring in terms of automatic concept adjustment, are the main novelties of this research, and EVPs are intended to have applicability beyond compositional classification. Subsequent sections in this chapter compare this work with other, more generally related lines of research in metareasoning and concept refinement.

8.1 Learning With Structured Representations

There is a myriad of work on learning that makes use of structured representations. In this section, we highlight and discuss some research that is particularly pertinent to the techniques we have developed in this work. For the purposes of comparison, we will situate each technique discussed in this section along two axes of variation – first, the degree to which the technique exploits prior knowledge of decompositional hierarchical structure, and second, the degree to which the technique exploits semantic pinning (alternatively, the degree of supervision) of nodes within the hierarchy. ANs make strong use of both kinds of background knowledge, so this is an interesting basis for comparison of techniques. Notice that these axes are not orthogonal – it is not possible to semantically pin the components of decomposition within a hierarchy that does not exist! The next subsection covers those structured learning techniques that

Table 6: A partial taxonomy of structured classification techniques.



		Use of Structure For Problem Decomp.	
		Strong	Weak
Use of Supervision at Internal Nodes	Strong, with explicit abstraction	ANs	
	Strong, without explicit abstraction	Hybrid ANs, TSB, Layered Learning, Structured Induction, BNs	
	Moderate	KBANN, EBNN	
	Weak		HMEs

fit well within the taxonomy suggested by these axes, and the following subsection covers those that do not fit as neatly.

8.1.1 Classification Learners using Decomposition

In this subsection, we discuss classification learners that use structured representations to decompose the overall problem into a set of sub-classification problems. Table 6 summarizes where these learning methods fit into the space defined by the two axes described at the head of this section.

Work on tree-structured bias (TSB) [97][115] is the most closely related to ANs and the problem domain of compositional classification used to test EVP-based learning. In systems that make use of tree structured bias, a concept hierarchy like those represented by ANs is used to limit the hypothesis space that must be searched by a learner. So, like ANs, there is a strong exploitation of tree structure under TSB. One of the contributions of experimentation with Augur is the application of the general idea of tree-structured bias in new settings, including the use of ML techniques that have not been combined with tree-structured bias in the past and application to non-synthetic problems. This research also moves beyond past work on TSB in several other directions, studying, for example, the effects of faulty knowledge structures on learning and expanding on theoretical results. More significantly, there are several *fundamental* differences between ANs and past work on tree-structured bias. First, TSB has dealt only with binary classifications at all nodes in the hierarchy, while ANs can deal with multivalue classifications. As noted above, the primary distinction is that TSB research does not have the concept of EVPs. Though this is true of all of the techniques discussed in this section, there are some comparisons worth drawing here. In lieu of EVPs, TSB learners instead rely on carefully constructed queries to the environment to learn the functions at internal nodes. This procedure can be construed as requiring a very specific kind of empirical verifiability for internal nodes – thus forcing a particular (and rather complex) form on the EVPs that a designer would write if applying TSB procedures within the AN framework. In particular, an EVP for an internal node in a hierarchical classifier can be written such that it makes a series of TSB-style queries to the environment to determine the correct value of the associated node during learning. Tadepalli and Russell [115] show how to simulate an oracle function at internal nodes in a TSB hierarchy using the queries their work requires. This procedure is exactly what would be placed within an EVP. Hence, like ANs, TSB exploits semantic pinning at all nodes within the hierarchy, though this pinning is more implicitly expressed by the structure of the hierarchy in conjunction with the queries assumed available and the procedure for their use. In the work described here, we take the stance that, in general, a broader set of queries to the environment may be possible. If this is the case, it will be more efficient to make use of the observations that most directly allow us to determine the value of an internal node when learning. In fact, the motivating example given by Tadepalli and Russell [115], concerning a credit-card domain, appears clearly to have a strong kind of direct empirical verifiability at internal nodes that could be exploited by an AN using very simple EVPs. Thus, past work on TSB can be seen as a specialization of the techniques described in this paper, where only a particular kind of query is supported by the learning environment. Note also that the requirement that *any* example can be obtained from the environment by the learner is a rather strong assumption which may not hold in domains where only limited training samples are available. ANs do not require this assumption to hold. AN research also moves beyond TSB by allowing the construction of hybrid learners, where semantic pinning is employed only at some subset of nodes within the classification hierarchy.

Layered learning [130] makes use of decomposition hierarchies to address large learning problems. In layered learning, each component's learner is trained in a tailored environment specific to the component. Our AN technique is more akin to what is called "coevolution" of components in work on layered learning, where multiple learners in the decomposition hierarchy are trained simultaneously in the actual target domain. However, in layered learning, genetic algorithms are used for training. This means that the structural credit assignment problem is addressed through trial and error, which will not provide the type of scalability characteristics we expect to achieve with a systematic approach to credit assignment. An additional distinction is that ANs focus on progressive abstraction, limiting the number of inputs to each component and ensuring a learning problem of manageable dimensionality at each component. In contrast, layered learning focuses on temporal abstraction, where components responsible for selection of abstract actions are not necessarily shielded from the need to consider many raw state features. And ANs also allow the use of arbitrary (in principle, heterogeneous) learners within each component. Layered learning makes use of both strong hierarchical knowledge and strong semantic knowledge at each node.

Like the basic form of layered learning, Shapiro's structured induction [100] makes use of a hierarchical knowledge structure for classification, and trains each element individually from the bottom up. Shapiro's technique is specifically tailored to aid with the process of information extraction from an expert in building an expert system. This work goes beyond Shapiro's by proposing methods to automatically adjust node semantics, performing end-to-end training of the hierarchies, admitting multiple types of supervised classification learners within nodes, and implementing mixed hierarchies, where the semantics of only a subset of nodes are known. Shapiro also describes a mechanism by which a knowledge hierarchy may produce a human-readable explanation of its reasoning. The work described in this thesis does not incorporate such a mechanism, but in principle such a feature could be added. Like layered learning, both strong hierarchical knowledge and strong semantic pinning are used in structured induction.

Knowledge-based ANNs [118] and Explanation-based NNs [78] both apply background knowledge in order to speed up learning in supervised classification problems. In KBANN learning, neural network structure and initialization are informed by background knowledge in the form of Horn clauses. Then, the network is trained using a standard method such as backpropagation. That is, credit assignment during learning is based on structural and numerical properties of the knowledge representation. In contrast, credit assignment over ANs is based on fixed semantic properties of the structural elements. These semantic properties are explicitly encoded as Empirical Verification Procedures that ground the knowledge contained within a structural element in terms of falsifiable predictions about the environment. Also notice that the result of learning is different. With ANs, the structural elements of knowledge retain known, explicitly specified meanings. With KBANN, there is no guarantee that structural elements of the neural network that results from training will have any particular or identifiable meaning. So both the considered hypothesis spaces and the nature of the search through those spaces is different in KBANN vs. AN learning. Beyond the restriction bias of requiring fixed semantics for intermediate nodes, there are also other advantages such as the potential for transfer of partial networks to new problems and inspectability of knowledge. KBANN does exploit structural background knowledge, as well as semantic background knowledge for the purposes of initialization. However, there is no semantic pinning maintained at nodes within a KBANN hierarchy during training.

In EBNN, a neural network is trained via the TangentProp algorithm. TangentProp works as backpropagation, however it is augmented with knowledge about the desired derivatives of the output function with respect to changes in the input values. EBNN finds the derivatives used as input to TangentProp on a per-example basis using provided background knowledge. This background knowledge is in the form of an approximate representation of the target function by a set of neural networks. The representation used for the domain theory is similar to an AN with ANNs at each node, but EBNN does not deal with learning over this representation, but rather learns while treating this information as fixed background knowledge. As in the discussion about KBANN above, notice that AN learning differs from EBNN in both representation and in the procedure for credit assignment. EBNN learning results in a trained neural network, where intermediate nodes are not guaranteed to have any identifiable interpretation. In contrast, the AN representation always maintains known, explicitly represented interpretations for all intermediate nodes. In a related point, TangentProp credit assignment distributes blame across network weights based on structural characteristics of the network, rather than based on analysis of fixed node interpretations as is the case in AN learning. EBNN makes use of both structural and semantic background knowledge, though its use of semantic knowledge at nodes within the classification structure is better described as influence than pinning – preference bias vs. restriction.

Hierarchical Mixture of Experts (HME) learning [60] trains multiple experts (learners) to solve the same problem and then combines their outputs via a series of gates in order to produce a result. By training both the experts and gates, the HME is able to learn complex decision boundaries. However, an identifiable interpretation of the purposes of the experts and gates is not guaranteed by the training algorithm. This differs from the AN technique, where a single learner addresses each learning task within a network and an analytical credit assignment algorithm that respects assigned node semantics is used. HME learning uses neither background knowledge of the structure of a problem decomposition, nor the capacity for direct supervision of subproblems.

Bayesian Networks (BNs) [88] represent joint probability distributions efficiently by making use of conditional independence relationships among features. On the other hand, Abstraction Networks capture progressive aggregation and abstraction into equivalence classes, culminating in abstraction into a desired classification. This distinction has practical implications for the methods that operate on Abstraction Networks. First, the credit assignment procedure for ANs differs from learning in Bayes nets. During AN learning, Empirical Verification Procedures must be invoked to determine whether a particular abstraction (intermediate equivalence classification) was accurate. When learning over a Bayes net, this is never required as the represented variables are expected to be directly observable, or are estimated (e.g. using EM). The fact that there is a level of abstraction between concepts represented at nodes in an AN and features directly observable in the environment is also a source of power for ANs. Next, because this level of abstraction is via an explicitly represented mechanism (the Empirical Verification Procedure), this abstraction can be directly operated upon by learning. This means that the number of distinctions made by a given AN node can be adjusted, increasing or decreasing the level of distinction made by a particular set of equivalence classes. Also, the specific division of actual world states into these equivalence classes can be directly operated upon, potentially changing the constitution of equivalence classes. Because Bayes nets do not deal with features in terms of such explicitly represented abstractions, this type of operation is not possible when learning over Bayes nets. Of course, one could manually tune the equivalence classes used at various nodes within a Bayes net (presumably because the human designer *is* able to understand the abstraction mechanism at work even though it is not explicitly encoded). However, we are speaking here of the *automatic* adjustment of these equivalence classes by the learner. Of course, it is also quite possible that one could apply EVPs to Bayes nets. This would imbue Bayes nets with an explicit representation of their features' abstraction from raw perception, and the automation of equivalence class tuning should also be possible for Bayes nets if this were done. We also provide some empirical comparisons between ANs and BNs in Appendix B, along with additional comments. Bayes nets in their basic form make full use of both structural background information (though they require a less rigid form on this structural information than do hierarchical classifiers) and supervision at individual nodes. Although individual BN nodes are supervised, they do not have an explicitly specified layer of abstraction sitting between raw state and values represented at nodes. This is what EVPs provide in this work. However, many variants of BNs have been produced that relax these assumptions.

8.1.2 Other Structured Learners

In this subsection, we discuss learning methods related to ANs that do make use of structure, but do not fit neatly within the taxonomy used within the last section. In each discussion, we do note the relationship of the technique to the taxonomy.

Work on structured matching [37] [12] also focuses on classification hierarchies that progressively aggregate and abstract from raw state features, culminating with the production of a desired top-level classification. Structured matching, unlike all of the other techniques discussed in this section, is *not* a learning method, but rather a static knowledge representation (and associated inference mechanism) used in expert systems. However, it is worth mentioning here as it underscores the power and widespread use of hierarchical classification methods within the knowledge-based AI community. This work demonstrates computational characteristics of representing classification knowledge hierarchically, including the cost of drawing inferences with knowledge represented in such a fashion, and formalizes a type of classification frequently used in AI systems. Chandrasekaran [43] [15] differentiated bottom-up structured matching with top-down taxonomic classification. He viewed both of them as Generic Tasks because of their ubiquity in AI [14] [13]. However, structured matching dealt only with knowledge engineered classification hierarchies, and did not consider learning. As there is no learning, it is not really sensible to situate structured matching along our hierarchy/semantic pinning axes. As a fixed knowledge representation, structured matching has maximum bias on any conceivable axis.

Explanation-based learning (EBL) [76] is a knowledge-based approach to reasoning and learning in which classification (but not necessarily prediction) is also important. However, as noted by Russell and Norvig [96], given a training example EBL "does not actually learn anything factually new from the instance. The agent could have derived the example from what it already knew, though that might have required an unreasonable amount of computation." The point here is that after learning, a system employing EBL is not capable of correctly solving any more problems than it was able to solve directly using provided background knowledge, although the computational expense may have been considerable. EBL instead makes the system more efficient at solving the kinds of problems that are input as examples by reencoding background knowledge appropriately. EBL produces processing templates that can be used to solve future problems similar to input examples more efficiently. That is, the task here is speedup learning. This is not the same task solved by AN learning. The state of knowledge after processing an example is different from the state of knowledge before processing an example. It is not simply a transformation in the form of knowledge. To see this notice that the set of examples consistent with a given AN are not the same as the set of examples consistent with an AN after modification through learning. Thus, the task in EVP-based learning *is* expanding the set of solvable problems. EBL does exploit both structural and semantic domain knowledge, but in a way that differs substantially from a hierarchical classification system such as ANs/Augur.

One view of a secondary contribution of this work is the development of a method for scalable classification learning through use of domain knowledge. Work on hierarchical RL [26] has a similar goal, but in a different problem setting and with the use of a different kind of domain knowledge. In hierarchical RL, or partial programming [69], procedural knowledge in the form of a hierarchy of temporally abstract actions is used, and action selection (which can of course be seen as a particular kind of classification) is the goal. This type of background knowledge is well-suited for an RL problem setting. In the scenario addressed in this work, knowledge about equivalence classes relevant to classification is more directly useful. In hierarchical RL, different senses of hierarchy are at play, and different techniques make different assumptions about background knowledge that is available. In some cases, the hierarchy of actions possible at each level is fixed, though this is not always the case. There is also a procedural hierarchy defined by the policy, which is the target of learning. Actions at each level have semantics that are defined by the associated reward functions.

Stacked Generalization [134] is a method that allows classification learners (called "generalizers" in this work) to be combined via a meta-level learner that learns to guess an answer based on the guesses returned from all the base-level learners. This work differs from AN learning in that each of the base level generalizers is trained over the same learning problem – each attempting to learn the same target function. In contrast, ANs break a learning problem into distinct subproblems, each of which is handled by a node in the network. Though there is structure in stacked generalization, the notion of problem decomposition does not exist here, and so our comparison of the degree of structural and semantic background knowledge used is not particularly applicable to this technique.

8.2 Metareasoning and Metaknowledge

EVP-based learning takes a reasoned, deliberative approach to learning, where an agent reflects upon itself and decides to effect changes in order to correct errors. ANs encode both object-level knowledge (the sum of the classification knowledge stored within the supervised learners within nodes) as well as metaknowledge (the EVPs that fix the semantics of the object-level knowledge). The process of reflecting upon one's own reasoning and altering or guiding that reasoning in a deliberative way is referred to as *metareasoning*, as discussed at length in Chapter 1.

The Meta-AQUA system [22] is an example of a system that selects among various available reasoning strategies (and also among potential learning goals, an example of goal-driven learning discussed in Section 8.3). In this system, it is specifically learning techniques that are chosen among, according to properties of the techniques and of the situation currently faced by the agent.

In some ways work on active learning [71] [117] can be viewed as metareasoning, and there is a relationship with EVP-based learning. In both this work and in active learning, the learner must make reasoned decisions about when it is worth obtaining a class label (which in this work includes class labels at intermediate nodes in the hierarchy). So in the most general sense, the research described in this thesis *is* a kind of active learning. The major difference, of course, is that we deal in this work with hierarchical classification, which active learning generally does not. Also, the concept of EVPs, which explicitly encode the connection between equivalence classes in the hierarchy with raw percepts, are not present in active learning. Finally, much (though not all) work on active learning deals with a problem setting known as *pool-based* learning, where there is a pool of unlabeled examples for the learner to select from in choosing the next example it will receive. In this work we have generally made a different assumption, that potential examples come in sequence. However, it seems probable that AN learning could be adapted to the pool-based setting. It is likely that techniques for query selection suggested by active learning may be useful for selecting queries for nodes in AN hierarchies in such settings. However, in hierarchical classification learning we would not only wish to consider local measures such as variance in example classification at a given node, but also the impact of that local uncertainty on the overall classification (the value produced at the root of the hierarchy). These extensions may constitute an interesting future direction for AN-based hierarchical classification learning.

8.2.1 Model-Based Self-Adaptation

EVP-based learning takes a particular view of the learning process, specifically that learning can be viewed as *self-adaptation* through a process of *self-diagnosis* and *self-repair*. The general diagnosis (credit assignment) problem has been characterized as a core problem in learning [75]. Samuel [98] first identified the problem in his work on checkers playing programs. The causal backtracing diagnostic technique described in Section 2.4 is inspired by classic AI techniques in which probe points (here, EVP executions) are systematically chosen to decrease the size of the *diagnostic* hypothesis space based upon evidence gathered so far. A good overview of AI diagnosis is given by Stefik [105]. Self-diagnosis is central to this work, motivating the semantic grounding of knowledge via falsifiable predictions about perception, which we encode in the form of Empirical Verification Procedures.

The process of self-diagnosis has frequently been facilitated by the use of self-models, explicit representations of self that can be operated over by a reasoning process in attempting to identify causes for (or localize) failures, as well as when deciding upon and implementing adaptations. For instance, Williams' work on immobots [132] imbues systems viewed as immobile robots with the ability to self-regulate and self-repair by making physical configuration changes when problems are detected. Self-models are used both to detect problems and to make decisions about how to avoid or correct them. Problem detection depends upon knowledge within the model that describes normal, expected operation. This knowledge can be seen as predictive, in that it describes expected experience when subsystems are operating as desired. However, though they reason over self-models, immobots cannot be seen as metareasoning in the true sense, as the models represent the physical configuration of the system and not its reasoning processes.

The REM [86] and Autognostic [110] systems also make use of self-models with predictive information about a system's intended functioning. These systems truly do engage in metareasoning, as the models that they use are not models of physical systems but rather of (portions of) their own reasoning processes. Both of these systems have the capability to recognize failures of reasoning and intervene either through configuration changes or hard modifications in order to correct errors. Once again, failure detection and localization is made possible through the inclusion of predictive information within the self-models. REM is also capable of proactive adaptation if provided a description of a new problem domain, through the use of the same self-model used for retrospective (failure-driven) adaptation.

REM in particular uses models coded in a language called Task-Method-Knowledge Language (TMKL) (see also Section 1.1.1). TMKL models of software systems are expressed in terms of tasks, methods, and knowledge. A *task* describes user intent in terms of a computational goal producing a specific result. Tasks encode functional information – the production of the intended result is the function of a computation. It is for this reason that the models specified in TMKL are *teleological* – the purpose of computational units is explicitly represented. A *method* is a unit of computation that produces a result in a specified manner. The *knowledge* portion of the model describes the different concepts and relations that tasks and methods in the model can use and affect as well as logical axioms and other inferencing information involving those concepts and relations. TMKL has been shown to be more expressive than Hierarchical Task Networks (HTNs) [31], as TMKL enables explicit representation of subgoals and multiple plans for achieving a goal. Hoang, Lee-Urban and Munoz-Avila [46] designed a game-playing agent in both TMKL and HTN and noted that TMKL provided control structures and other features beyond those available in HTN, and that TMKL provides strictly more expressive power than HTNs. Figure 4 displays only the tasks (rectangles) and methods (rounded rectangles) of the FreeCiv playing agent.

EVP-based learning is not metareasoning in the true sense, as it involves reasoning about domain knowledge rather than about reasoning itself. However, the notion that diagnosis is facilitated by the use of explicitly represented models (here, metaknowledge, as EVPs encode knowledge about knowledge) that contain predictive information is at the heart of the EVP method. These notions of deliberative self-diagnosis are also important in some of the techniques for concept refinement discussed in the next section.

8.3 Learning What to Learn: Refining Concept Semantics

Like the research discussed in the previous section, work on goal-driven learning [92] [64] [22] takes a strongly deliberative approach to learning. In goal-driven learning, a system dynamically determines which concepts are currently important to the system so that it may focus resources on particular learning tasks within a practically unbounded set of potential learning tasks in a rich environment. This process may also be one of diagnosis, though in goal-driven systems the learning goal selection process may be more proactive than retrospective. Goal-driven learning recognizes that knowledge does not gain value in a vacuum, but rather derives its value from its ability to contribute to the system's larger goals. In this sense, EVP-based learning can be seen as a kind of goal-driven learning, as decisions to alter concept semantics (Chapter 6) are made based upon each concept's observed ability to contribute to overarching goals. While goal-driven learning does not necessarily directly alter the semantics of concepts that the agent considers learning, the idea of knowledge having value in a goal context is an important one in EVP-based learning.

Work on Probabilistic Concept Hierarchies [32][50] makes use of knowledge representations centered around the generation of progressively more abstract equivalence classes, as do ANs. Probabilistic Concept Hierarchies, however, are used for unsupervised concept learning. This stands in contrast to our approach, where the usefulness of a particular concept is evaluated in terms of a specific higher level goal of the agent. In our work, this translates to the existence of a well-defined top-level target concept (classification) that is to be learned. Thus, relevance of information and usefulness of specific equivalence classes can be defined directly in terms of the target concept. In unsupervised learning, the goal can be thought of as forming equivalence classes that explain the data. Thus, in unsupervised learning, equivalence classes are valued based on how well they summarize available data. On the other hand, when learning a target top-level concept, the usefulness of an equivalence class is based on its ability to discard information irrelevant to the target concept.

Work on Predictive State Representations (PSRs) [67] [102] outlines rationale for using state representations that directly encode predictions about future events. In a general way, the notion that knowledge should have a predictive interpretation is central to this work as well, as noted in the previous section. The requirement for empirical determinability is a requirement that knowledge make a verifiable prediction. In work on learning PSRs [131] [135], the system can be seen as defining concepts relevant to action selection, as the predictive representations of state are constructed on-line. However, the tasks addressed with PSRs vs. EVP-based learning are substantially different. While PSRs are used to build state representations based upon predicted action/observation sequences, ANs are used to encode and learn hierarchical classification knowledge. For this reason, work on PSRs has not focused on the problems of self-diagnosis, self-repair and bias in inductive learning that are central to this work.

The information bottleneck method [116] provides a means by which one may find an optimal compression of a signal that maintains a given amount of information about another random variable. Classification is a perspective on compression, and the generalization power afforded by the AN hierarchical representation is due to lossy compression at each node within the hierarchy – thus, each node is performing a form of information bottlenecking, though the method used is not the same as that of Tishby, Pereira and Bialek. When EVP adjustment is triggered during EVP-based AN learning, we are dynamically adjusting (in this work, increasing) the amount of information we require to be passed through the associated AN node. Thus, the learning procedure with EVP adjustment is progressively estimating the appropriate degree of information loss at each node within the classification hierarchy.

Ivanov and Blumberg's work on clustering under reward [51] also takes the view that concepts have value based upon their ability to support action selection. In this work, clusters (concepts) are learned over perceptual input based upon their ability to enable the expression of a good action selection policy. Of course, the resulting concepts may not have the same human-identifiable semantics available when using EVP-based learning. Also, once again the problem setting and technical approach is substantially different from that in EVP-based learning. However, the general insight that knowledge derives value from the support of action selection is also important here. McCallum, in his work on utile distinctions [72] makes a similar observation. In this work, a system learns to represent state with varying amounts of history. The amount of history kept for a particular state is selected based upon the system's need to distinguish states in which different actions are preferred (the learned state space is used in support of reinforcement learning). Both of these lines of research share with this thesis the view that knowledge ultimately derives value from its capacity to support action selection.

CHAPTER IX

CONCLUSION

9.1 Summary

The work described in this dissertation has produced a number of technical contributions and gives rise to several claims which are supported through empirical and/or formal means. Before turning to a more detailed discussion of these claims and the broader implications of the work, we begin by summarizing these things.

Technical contributions:

- A theory of a type of metaknowledge that is useful for reflection upon and adaptation of domain knowledge, Empirical Verification Procedures. (Dissertation as a whole).
- A design, including algorithms and data structures, for the application of Empirical Verification Procedures within the context of compositional classification problems. (Chapter 2).
- A set of formal results illustrating various aspects of learning in compositional classification problems. (Chapter 7).
- An implementation, realized in the Augur system, of an EVP-based learning system applicable to compositional classification problems.

Claims:

- Empirical Verification Procedures allow a meta-level process to diagnose and repair faults in compositional classification knowledge stored within an Abstraction Network. (Chapter 4). The effectiveness of EVPs in this regard is neither specific to the learner type used within the AN (Section 4.1) nor to a particular compositional classification problem instance (Section 4.2). This effectiveness also stands in the face of hierarchies built with suboptimal knowledge engineering. (Section 4.3).

- Empirical Verification Procedures associated with nodes in a classification hierarchy impose a restriction bias on the hypotheses representable by that knowledge structure, and therefore increase the speed and accuracy of learning when this bias is correct. (Chapters 4-5).

- When Empirical Verification Procedures are not available for some of the nodes within an Abstraction Network, training is slower and more difficult than when using an AN with EVPs at all nodes. However, "incomplete" ANs in which a significant fraction of nodes lack EVPs still learn more rapidly than unstructured learners. Thus, ANs should be employed even in cases where EVPs are not available at all nodes. (Chapter 5).

- Concepts with semantics explicitly expressed through the use of Empirical Verification Procedures can be successfully automatically adjusted when they are inappropriately defined with respect to fulfilling their functional role in a system, at least when the EVPs are *quantizing EVPs*, which were the type adjusted in the experiments described in Chapter 6.

- Hierarchical classification knowledge structures introduce substantial restriction bias to a learner's hypothesis space, allowing more generalization to occur and thus a larger decrease in error rate per example. (Sections 4.1 & 7.3).

- The performance of learners using hierarchical classification knowledge structures degrades gracefully when they are built with structural errors. (Section 4.3).

- When applicable to the learner type used within nodes in an Abstraction Network and when the cost of EVP execution relative to example acquisition cost is high, a causal backtracing style of diagnosis is desirable when errors must be both *subjective* (propagated forward in the decision structure) and *objective* (wrong according to perceptual predictions). (Section 7.1).

9.2 Compositional Classification

Before turning to the central contributions of this dissertation, we will begin by discussing in more detail a few secondary contributions that arise due to the evaluation of the central hypothesis within the context of compositional classification. These results provide additional understanding of the compositional classification problem and the hierarchical knowledge representations (ANs) used to solve such problems. One of these contributions is the quantification of the hypothesis space reduction afforded by hierarchical classification knowledge representations, as well as empirical illustrations of the learning speed increase afforded by this structural restriction bias (Section 7.3 and Chapter 4). By connecting the hypotheses expressible using classification hierarchies with a well-studied mathematical construct, partition lattices, a path for developing a deeper understanding of the properties of these knowledge structures is laid.

We also present a demonstration of the optimality of the causal backtracing style of diagnosis (Table 2), along with a discussion of when the technique may lead to convergence problems – specifically, when learners within the classification hierarchy require that examples be drawn from a stable distribution. In fact, based upon the experiments and analyses described here, a design space emerges, describing the parameters that must be selected when applying hierarchical knowledge representations to learning in compositional classification problems, summarized in Table 7.

Table 7 shows when one is likely to prefer (or require) complete execution of all

Table 7: Design space for Abstraction Networks.

		Cost of EVP Execution Relative to Examples	
		High	Low
Learner Requires Stable Distribution?	Yes	Full EVP Exec.	Full EVP Exec.
	No	Causal Backtracing EVP Exec.	Full EVP Exec.

EVPs within an Abstraction Network, based upon whether the learner type chosen requires that examples be drawn from a stable distribution, as well as whether EVP execution cost has a relatively low or high priority relative to the cost of obtaining examples. EVP vs. example cost is relevant because causal backtracing diagnosis is parsimonious with respect to EVP execution (Section 7.1) at the cost of possibly not performing learning at all nodes where it may be possible for each example. These relative costs will be dictated by the domain. The choice of learner type will also be tied to problem specifics, based upon the type of inductive bias that is most appropriate for the various sub-classification problems induced by the knowledge hierarchy.

We have also presented empirical results of the impact of degraded knowledge engineering on the effectiveness of learning using an Abstraction Network (Section 4.3). These experiments show a graceful degradation of the performance of an AN-based learner as increasingly severe deficits in knowledge engineering are introduced. When no input information is lost, this degradation takes the form of decreased learning speed due to a reduced restriction bias. When input information is lost, the final error rate reached is also higher. However, the general finding is that use of structural information is substantially beneficial even if it is significantly incomplete.

9.3 Empirical Verification Procedures

The central claim that we make in this dissertation is that explicit semantic grounding of domain knowledge in perception makes it possible to self-diagnose and repair errors within that domain knowledge. Further, we claim that this semantic grounding in perception constrains the expressivity of the concepts that form the domain knowledge. These constraints form a restriction bias when learning over knowledge structures containing the concepts, and thus increase generalization from training examples. The experiments of Chapters 4 & 5 and the formal analysis of Section 7.2 support this claim. The experiments demonstrate the general usefulness of Empirical Verification Procedures that ground knowledge in perception across problem instances and learner types within compositional classification, and the gains in learning efficiency that can be attributed to the use of EVPs within hierarchical classification structures. From a broader perspective, the theme of this dissertation can be seen as making the following statement about knowledge:

> **Knowledge has meaning because it entails predictions about perceptions.**

Because of this stance, the work described here can be seen as implementing a *correspondence theory of truth*. The value of this stance from a purely pragmatic perspective is that a system with correspondence-based justifications of its knowledge is able to automatically update and refine that knowledge in order to increase its accuracy – where accuracy is judged by the ability of the knowledge to entail accurate inferences about the environment. As this is the metric by which we typically choose to judge the correctness of a system's knowledge, it is natural to embed this criterion in the mechanism of knowledge acquisition. Because the Empirical Verification Procedures that encode the grounding of knowledge in perception are pieces of knowledge about knowledge, they are properly termed metaknowledge. This work contributes to the field of metareasoning by providing a representation that allows reflection to move beyond self-diagnosis of faults in an agent's process to include faults in an agent's knowledge.

As mentioned above, fixing the semantics of concepts in perception by using EVPs constrains the expressivity of those concepts, as they are free only to express knowledge consistent with the perceptual expectations to which they are tied. Chapter 5 describes a set of experiments in which we remove the EVPs from some of the nodes within an Abstraction Network before training. The consistent result of these ablations is that learning performance in terms of generalization per example is decreased. These experiments demonstrate the restriction bias imposed by EVPs themselves, beyond the effects of knowledge structure. Because of this restriction bias, concepts within an overall knowledge structure that are semantically “pinned” by EVPs are learned more quickly (in terms of the number of examples required to learn the concept) than those that are not. This finding demonstrates that, from the perspective of learning efficiency, it is preferable to ground concepts directly in perception when possible, rather than solely through connection to other concepts.

An additional benefit of explicitly representing the semantic grounding of knowledge in perception is that this semantic grounding itself becomes a first class object that can be automatically operated upon by the agent. These ideas are described and their usefulness empirically supported in Chapter 6. Given that an agent has the capability to alter the sets of equivalence classes into which it abstracts perceived scenarios, a question is immediately raised: upon what basis should an agent decide to alter the semantics of one of its concepts? Or, to state the question positively, what makes a particular abstraction useful? The answer given by this research leads to another statement that this work makes about knowledge:

> **A concept’s value ultimately stems from its ability to support action selection.**

In this dissertation, the ultimate ability to support action selection translates to support of the production of a correct top-level classification. The target concept of this top-level classification is fixed, in this work, by some external force that asserts its meaningfulness. Consider the possible motives of the external force, though these motives are not explicit in the work described here. In each of the three non-synthetic domains discussed here, the ultimate goal must be action selection – in FreeCiv, one wishes to build a city in a desirable location; in DJIA prediction, one wishes to invest wisely; and in sports prediction one wishes to place a winning bet. If a concept does not in some way contribute to the production of behavior, it is not clear that there is a basis to consider that concept useful in any practical sense. If one does nothing (and cannot intend to do anything) with an inference, what can the value of that inference be? Thus, as the *constitution* of an equivalence class is justified based on grounding in predictions about perceptions (enforcing a given meaning), the *meaning* of an equivalence class is itself ultimately justified based on predictions about relative action outcomes. In the implementation discussed in this thesis, when the first kind of justification is violated, we change knowledge stored within a node in the AN to alter the constitution of an equivalence class. When the second kind of justification is violated (a parent node of a node in question is not properly supported in its function by its children), we alter the node's EVP in order to alter the intended meaning of the set of equivalence classes produced by the node.

This view of meaning and the value of knowledge leads directly to a particular view of error within classification hierarchies, where we see the need to alter knowledge at a node only if it is both *objectively* incorrect, based upon violation of the perceptual predictions it entails (EVP violation) and *subjectively* incorrect, based recursively upon the existence of an error at the parent node. That is, the knowledge in question must both contradict perception and fail to fulfill its functional role in the overall structure in which it exists. This view of error, then, leads to the "causal backtracing" style of diagnosis described in Section 2.4. This thesis also formally demonstrates the optimality of this diagnostic procedure under a set of assumptions in Section 7.1, and presents constraints on the applicability of the procedure – specifically, that learners used within AN nodes must not depend upon training examples being drawn from a stable distribution if this type of diagnostic procedure is to be used. This limitation means that we must sometimes adhere to a strictly objective view of error in our diagnostic procedure, in order to serve the technical needs of learner types operating within a classification hierarchy. If the subjective + objective view of error is seen as more desirable, this will have bearing upon the types of supervised learning methods that a designer opts to use within a knowledge structure.

Seen from the broadest perspective, a contribution of this dissertation is one of perspective and problem framing. In particular, this research views learning agents as fully situated, gaining information through rich interaction with the environment. In the context of classification learning, this means that we do not anticipate a narrow, fixed interaction with the environment where some set of training examples is pushed through a mailslot to an agent that sits in the corner of a windowless room learning to make predictions. Rather, we expect that an agent learning to make classifications is doing so in the interest of action selection, and is actively engaged with the environment about which it is attempting to learn. This means that, when failure is encountered, the agent can, over time, actively attempt to gather information about the causes of the failure in a manner reasoned out through reflection. In some problem settings this view may be inappropriate, if all that is truly available for learning is some set of examples. However, if the learning agent is able to interact more fully with its environment, the learning problem can be made much easier. Though some research such as goal-driven learning [92] and active learning [71] [117] does take a more situated approach to learning, this work goes further by imbuing the agent with an understanding of the meaning of its own knowledge in terms of observation and action.

APPENDIX A

COMPUTATION OF EXPRESSIBLE HYPOTHESES IN A SIMPLE HIERARCHY

This appendix details the manual computation of the number of hypotheses expressible by the hierarchical classifier of Figure 37. We begin from the allowable starting partitions from the stage 1 classifier:

> 14/23/58/67, 1/234/5/678, 124/3/568/7, 13/24/57/68, 123/4/567/8, 134/2/578/6, 12/34/56/78

Because only two class labels will be applied at the output of the stage 2 (top-level) classifier, we know that we only need to compute partitions consisting of two sets, and also count the single partition consisting of one set. We also know that there are seven unique ways to partition four items into two sets:

> 14/23, 1/234, 124/3, 13/24, 123/4, 134/2, 12/34

So, for each of our starting points (each of which consists of four sets), we now enumerate the results of making each of these 7 partitioning choices:

- 14/23/58/67
    - 1467/2358
    - 14/235678
    - 123467/58
    - 1458/2367

  - 123458/67
  - 145678/23
  - 1234/5678

- 1/234/5/678
  - 1678/2345
  - 1/2345678
  - 1234678/5
  - 15/234678
  - 12345/678
  - 15678/234
  - 1234/5678

- 124/3/568/7
  - 1247/3568
  - 124/35678
  - 12347/568
  - 124568/37
  - 1234568/7
  - 1245678/3
  - 1234/5678

- 13/24/57/68
  - 1368/2457

    - 13/245678
    - 123468/57
    - 1357/2468
    - 123457/68
    - 135678/24
    - 1234/5678

- 123/4/567/8
    - 1238/4567
    - 123/45678
    - 12348/567
    - 123567/48
    - 1234567/8
    - 1235678/4
    - 1234/5678

- 134/2/578/6
    - 1346/2578
    - 134/25678
    - 12346/578
    - 134578/26
    - 1234578/6
    - 1345678/2
    - 1234/5678

- 12/34/56/78
  - 1278/3456
  - 12/345678
  - 123478/56
  - 1256/3478
  - 123456/78
  - 125678/34
  - 1234/5678

There is only one point of overlap, the partition 1234/5678, which occurs as a result of applying the partitioning choice 12/34 at each of the 7 starting points within the 8 item lattice. This gives us:

> 6 unique partitions from each of 7 possible start points = 42 partitions +
> 1 repeated partition (1234/5678) = 43 partitions of size 2
>
> 1 partition of size 1 (12345678)
>
> So, expressible hypotheses: $43(2)_2 + 1(2)_1 = 88$
>
> Without structure: $2^8 = 256$ expressible hypotheses.
>
> For a savings of $256 - 88 = 168$ excluded hypotheses, a 65.6% reduction in hypothesis space size.

APPENDIX B

EMPIRICAL COMPARISON WITH BAYES NETS

In this appendix, we describe an empirical comparison between the performance of Bayesian Networks (BNs) and ANs. While these experiments have no bearing upon the claims made in this dissertation, as EVP-based reflective learning could be used equally well within the context of BNs as ANs, the results are somewhat interesting in their own right and so are included here for the sake of completeness.

We used a randomly generated set of synthetic learning problems (see Section 4.1) to compare the performance of ANs with Bayesian Networks (BNs). The environment consists of a fixed abstraction network, over which no learning will occur, that represents the correct, target content (and structure) for the problem. Given this fixed AN, we then create a separate “learner” AN that will be initialized with random knowledge content and expected to learn to functionally match the content of the target AN. We also create a BN with identical structure and initialize the CPTs randomly. We used the Bayes Net Toolbox (BNT) implementation of BNs, learning with sequential parameter updating from complete data. Note that this means that the BN examines the true value of every feature when learning from each example, while the AN learner in general does not do so. This experiment did use the causal backtracing style of diagnosis described in Section 2.4, and thus the AN learner did not examine the proper value of each node for each training example. Because the work described here is concerned only with repairing content and not structure, we do build the learners with correct structure that matches that of the fixed AN. Training proceeds by (1) generating a pseudo-random sequence of floating point numbers to serve as the observations for the input nodes of the ANs, (2) performing inference

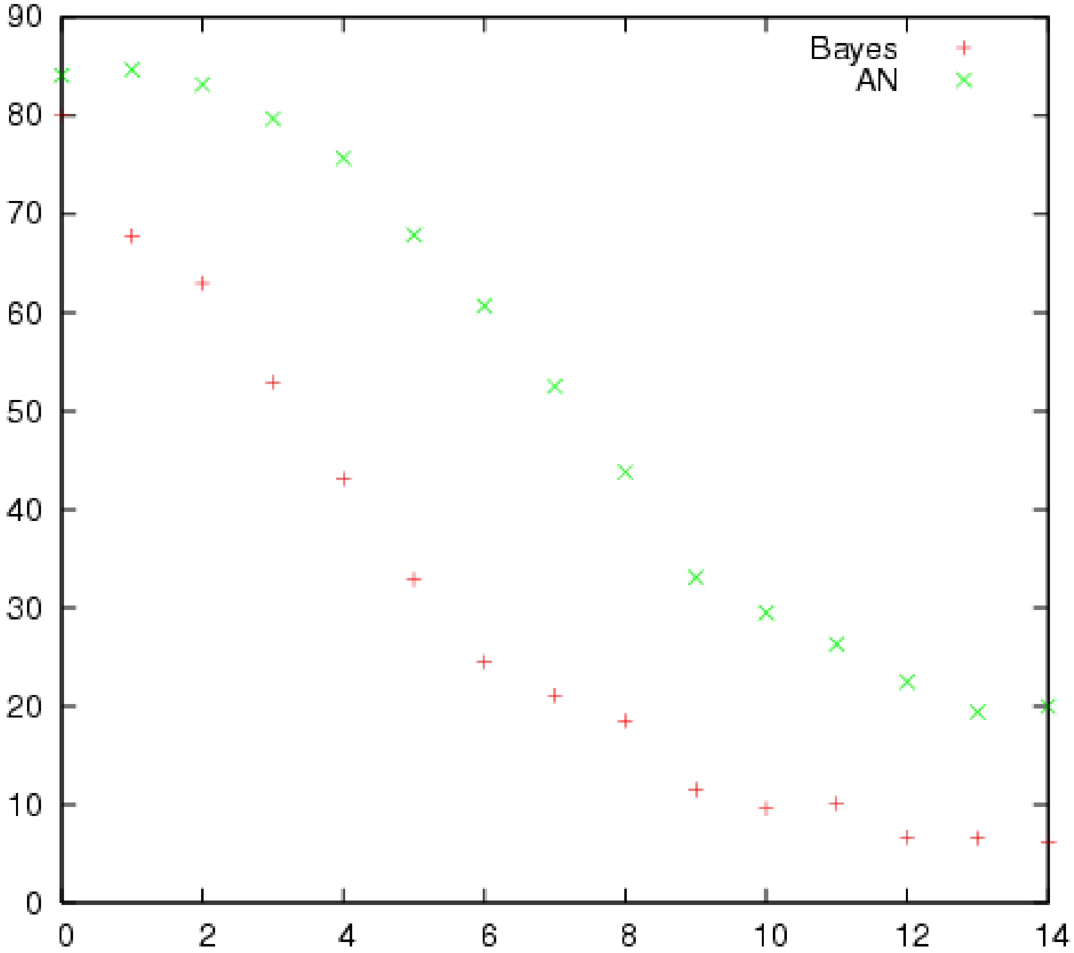


Figure 38: Error per block (Y-axis) vs. block number (X-axis) for network layer sizes 10-5-2-1, 6 values per node, 100 examples per block, averaged over 15 random repeats.

with the fixed AN, saving the values produced by all intermediate nodes as well as the root node, (3) performing inference with the AN and BN learners and (4) performing EVP-based self-diagnosis and learning over the AN and BN learners according to the procedures described above. EVPs within the inputs of both ANs are set up to quantize the floating point observations. EVPs are not needed at non-leaf nodes in the fixed AN, since no learning will occur. EVPs at non-leaf nodes in the learning AN are set up to examine the saved output value from the corresponding node in the fixed AN, while the output values from all nodes in the fixed AN are composed into a

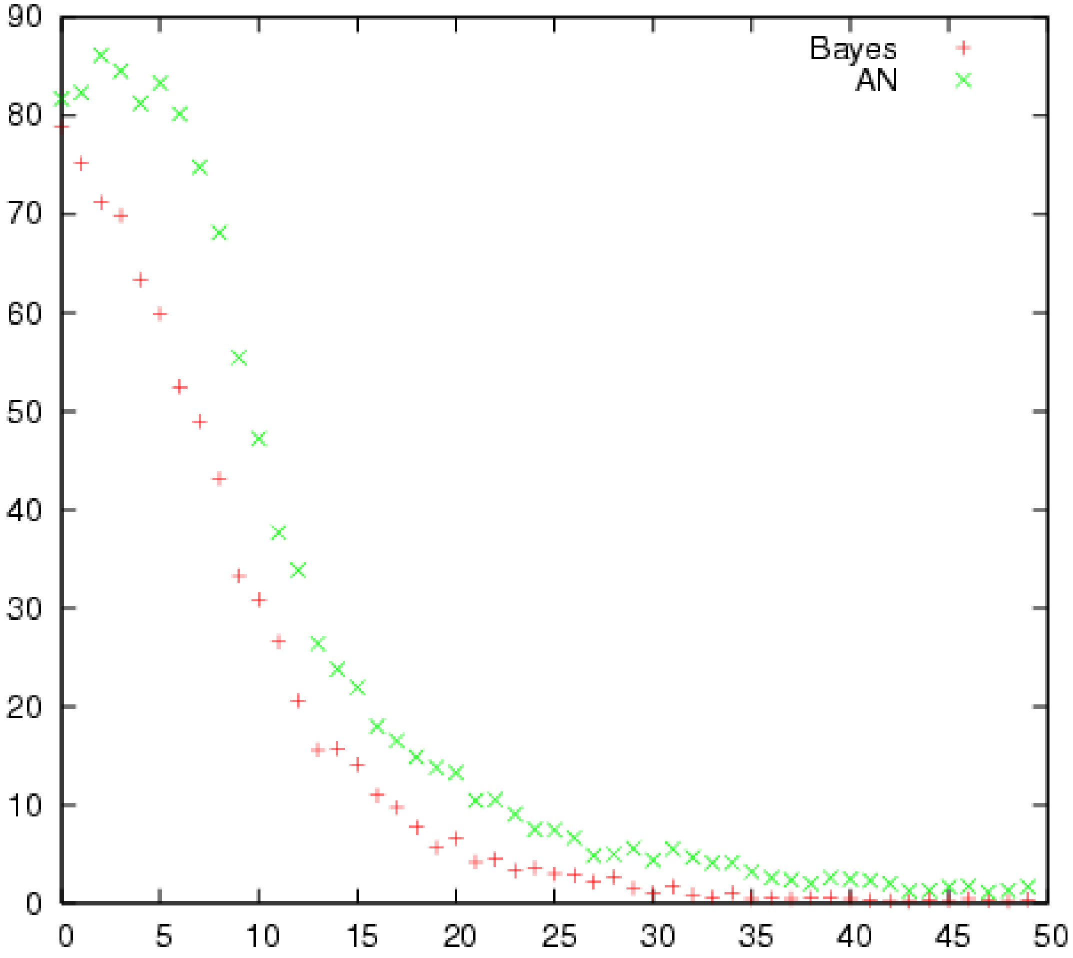


Figure 39: Error per block (Y-axis) vs. block number (X-axis) for network layer sizes 20-10-5-2-1, 6 values per node, 100 examples per block, averaged over 15 random repeats.

complete feature vector for use by the BN learner. In these experiments we once again use simple rote learners within each node in the learner AN. Results obtained when inputs were drawn from a non-uniform distribution are depicted in Figures 38-40.

These results indicate that ANs offer competitive learning per example as problem size increases in at least some learning scenarios, even though fewer feature values are examined per example. For instance, in the scenario depicted in Figure 39, the AN learner examined an average of 6.98 feature values per example, while the BN learner examined all 8 non-leaf values for each example. If the resource cost of obtaining

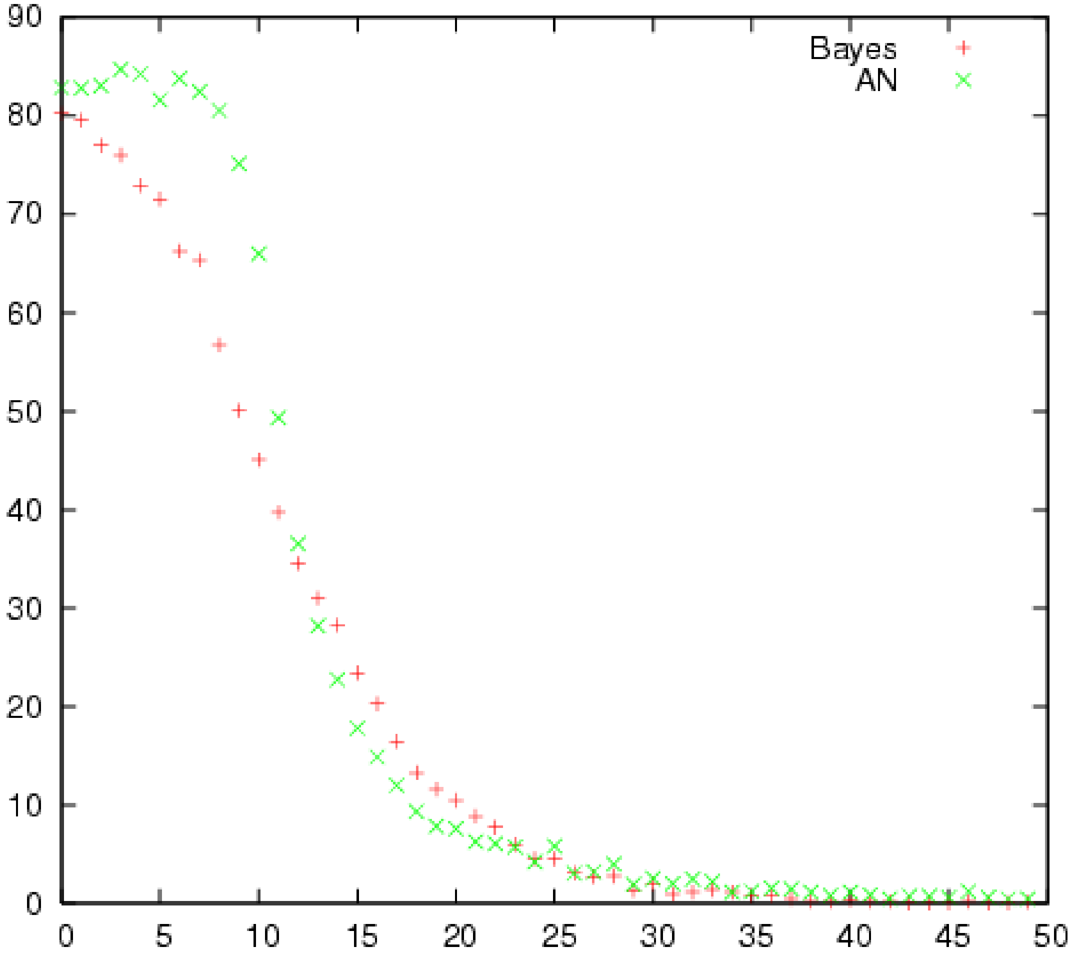


Figure 40: Error per block (Y-axis) vs. block number (X-axis) for network layer sizes 32-16-8-4-2-1, 6 values per node, 100 examples per block, averaged over 15 random repeats.

feature values from the environment is significant, this property of ANs translates to resource (e.g. time) savings. However, the AN still gave comparable to better performance in terms of error rate decrease per example. In particular, it is interesting to note that the performance of the AN learner approaches and finally surpasses that of the BN learner as problem size increases from Figure 38 through Figure 40. The initial flat stage for the AN learner, most noticeable in the second two scenarios, is likely due to the need to progressively learn at each of the layers in the AN before an overall decrease in errors is realized.

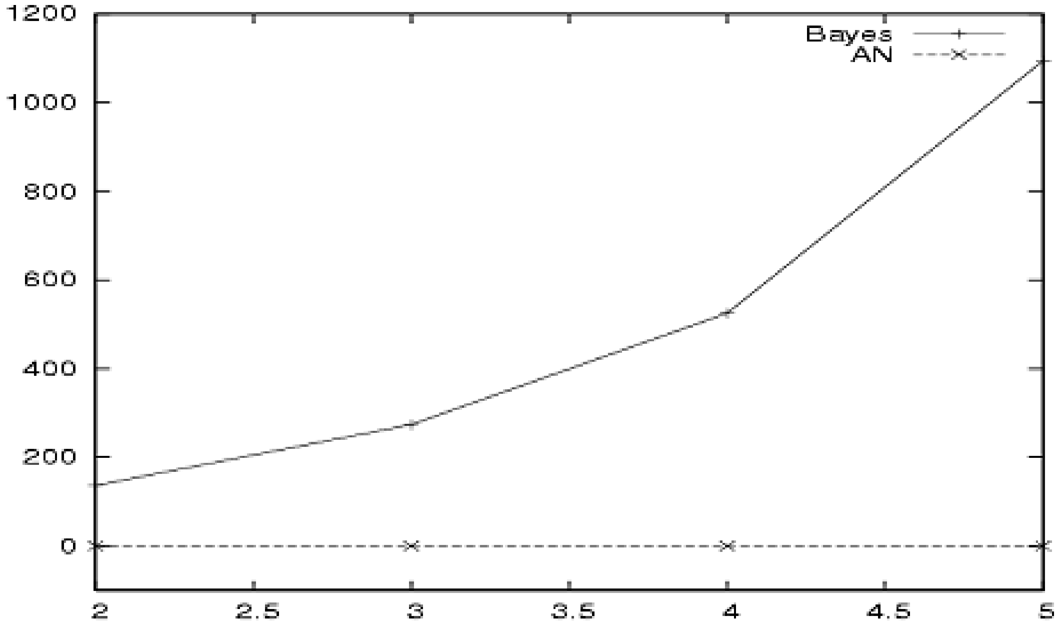


Figure 41: # of layers in binary tree vs time with 5 values possible at each node, 10 blocks of 10 examples.

The time cost of online learning and inference in ANs vs BNs as a function of network size is show in Figure 41. Here, time is shown on the Y-axis, and network size is shown on the X-axis. The X-axis values are the number of layers in the network trained, and each network is a binary tree (thus, for example, a 3-layer tree in this experiment will have 7 nodes).

Empirically, it appears that the cost of online learning and inference for BNs is increasing more rapidly than that of ANs as a function of network size, though it is not clear to what extent this difference is due to implementation specifics. An additional note here is that probing of actual variable values was done once, and the results provided to both the AN and BN learners (though only results requested by the AN self-diagnosis procedure would actually be passed to the AN). This means that the time cost of the additional probes required for the BN learner is not reflected in the data presented in this section. If the procedures for obtaining variable values have non-negligible cost, the time used by BNs will in practice be even more significantly above that used by an AN. Of course, BNs are more general than ANs, but it does appear that there is some advantage in problems where ANs are applicable. It is likely that these advantages are due to the fact that ANs pass less information from node to node during inference (ANs commit, in effect, to the most likely value at each node, while BNs pass a distribution over all values). It is clear why this difference between ANs and classical BNs would lead to a difference in computational effort during inference, and it seems likely that observed differences in learning rate are similarly attributable. Thus, one could likely match the performance of ANs in these experiments by using BNs that commit fully to the most likely value at each node during inference (setting the probability of the most likely value to 1 and all other values to 0). However, once again this experiment is quite peripheral to the thesis here, as EVPs could equally well be used in BN learning, providing advantages such as automatic concept refinement within the context of BN learning.

REFERENCES

[1] "Leading indicators of employment," *Australian Economic Indicators*, vol. 1350.0, April 2004.

[2] "RBC US leading economic indicator," *Economics Department, RBC Financial Group*, November 2005.

[3] Agogino, A. and Tumer, K., "Unifying temporal and structural credit assignment problems.," in *AAMAS*, pp. 980–987, 2004.

[4] Anderson, M. L. and Oates, T., "A review of recent research in metareasoning and metalearning," *AI Magazine*, vol. 28, pp. 7–16, 2007.

[5] Anderson, M. L., Oates, T., Chong, W., and Perlis, D., "The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance," *J. Exp. Theor. Artif. Intell.*, vol. 18, no. 3, pp. 387–411, 2006.

[6] B. Krulwich, L. B. and Collins, G., "Learning several lessons from one experience," in *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, pp. 242–247, 1992.

[7] Barto, A. G. and Mahadevan, S., "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[8] Bennet, J. and Engelmore, R., "SACON: A knowledge-based consultant for structural analysis," *Proc. Sixth International Joint Conference on Artificial Intelligence*, pp. 279–284, 1979.

[9] Birnbaum, L., Collins, G., Freed, M., and Krulwich, B., "Model-based diagnosis of planning failures," *In Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 318–323, 1990.

[10] Brachman, R., "Systems that know what they are doing," *IEEE Intelligent Systems*, pp. 67–71, Nov/Dec 2002.

[11] Bylander, T. and Chandrasekaran, B., "Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition," *Int. J. Man-Mach. Stud.*, vol. 26, no. 2, pp. 231–243, 1987.

[12] Bylander, T., Johnson, T., and Goel, A., "Structured matching: a task-specific technique for making decisions," *Knowledge Acquisition*, vol. 3, pp. 1–20, 1991.

[13] Chandrasekaran, B., "Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples," in *The Knowledge Engineering Review*, vol. 3, pp. 183–210, 1988.

[14] Chandrasekaran, B., "Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design," pp. 170–177, 1993.

[15] Chandrasekaran, B. and Goel, A., "From numbers to symbols to knowledge structures: Artificial Intelligence perspectives on the classification task," *IEEE Trans. Systems, Man & Cybernetics*, vol. 18, pp. 415–424, May/June 1988.

[16] Chandrasekaran, B. and Mittal, S., "Conceptual representation of medical knowledge for diagnosis by computer: Mdx and related systems," in *Advances in Computers: Volume 22* (Yovits, M., ed.), pp. 217–293, Academic Press, 1983.

[17] Chapin, N., Hale, J. E., Kham, K. M., Ramil, J. F., and Tan, W.-G., "Types of software evolution and software maintenance," *Journal of Software Maintenance*, vol. 13, no. 1, pp. 3–30, 2001.

[18] Clancey, W. J., "Heuristic classification.," *Artif. Intell.*, vol. 27, no. 3, pp. 289–350, 1985.

[19] Cox, M. T. and Raja, A., "Metareasoning: A Manifesto, Technical Report, BBN TM-2028, BBN Technologies," 2007.

[20] Cox, M. T., "Metacognition in computation: A selected research review," *Artif. Intell.*, vol. 169, no. 2, pp. 104–141, 2005.

[21] Cox, M. T., "Metareasoning, monitoring, and self-explanation," in *AAMAS-07 Workshop on Metareasoning*, 2007.

[22] Cox, M. and Ram, A., "Introspective multistrategy learning: On the construction of learning strategies," *Artificial Intelligence*, vol. 112, pp. 1–55, 1999.

[23] Davis, R., "Interactive transfer of expertise: Acquisition of new inference rules," *Artificial Intelligence*, vol. 12(2), pp. 121–157, 1979.

[24] Davis, R., "Meta-rules: Reasoning about control," *Artificial Intelligence*, vol. 15, pp. 179–222, 1980.

[25] Dietterich, T., "An overview of MAXQ hierarchical reinforcement learning," *Lecture Notes in Computer Science*, vol. 1864, 2000.

[26] Dietterich, T. G., "The MAXQ method for hierarchical reinforcement learning," in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 118–126, 1998.

[27] Doyle, J., "A truth maintenance system," *Artificial Intelligence*, vol. 12, pp. 231–272, 1979.

[28] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern classification and scene analysis.* Wiley New York, 1973.

[29] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification, Second Edition.* New York: Wiley, 2000.

[30] Duda, R., Hart, P., Barret, P., Gasching, J., Konolige, K., and Reboh, R., "Development of the Prospector consultation system for mineral exploration," *SRI International Menlo Park, CA, technical report*, 1979.

[31] Erol, K., Hendler, J., and Nau, D. S., "HTN planning: Complexity and expressivity," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, vol. 2, (Seattle, Washington, USA), pp. 1123–1128, AAAI Press/MIT Press, 1994.

[32] Fisher, D. H., "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, 1987.

[33] Fox, S. and Leake, D., "Introspective reasoning for index refinement in case-based reasoning," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 13, pp. 63–88, 2001.

[34] Fox, S. and Leake, D. B., "Using introspective reasoning to refine indexing," in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 1995.

[35] Ganek, A. G. and Corbi, T. A., "The dawning of the autonomic computing era," *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, 2003.

[36] Genesereth, M., "An overview of meta-level architecture," *In Proceedings of the Third National Conference on Artificial Intelligence*, pp. 119–123, 1983.

[37] Goel, A. and Bylander, T., "Computational feasibility of structured matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 12, pp. 1312–1316, 1989.

[38] Goel, A., Jones, J., Parnin, C., Rugaber, S., and Sinharoy, A., "Teleological reasoning in software adaptation: A case study in game playing agents," *Third International Conference on Design Science Research in Information Systems and Technology (DESRIST-2008)*, pp. 365–379, May 2008.

[39] Goel, A., Morse, E., Raja, A., Scholtz, J., and Stasko, J., "Introspective self-explanations for report generation in intelligence analysis," *IJCAI-09 Workshop on Explanation-Aware Computing*, July 2009.

[40] Goel, A., Stroulia, E., Chen, Z., and Rowland, P., "Model-based reconfiguration of schema-based reactive control architectures," in *In Proceedings of the AAAI Fall Symposium on Model-Based Autonomy*, pp. 23–29, Morgan Kaufman Publishers, 1997.

[41] Goel, A. K., Gomez, A., Grue, N., Murdock, W., Recker, M., and Govindaraj, T., "Explanatory interface in interactive design environments," *Fourth International Conference on Artificial Intelligence in Design (AID-96)*, 1996.

[42] Goel, A. K. and Murdock, J. W., "Meta-cases: Explaining case-based reasoning," *European Workshop on Case-Based Reasoning (EWCBR)*, pp. 150–163, 1996.

[43] Goel, A. K., Soundararajan, N., and Chandrasekaran, B., "Complexity in classificatory reasoning," in *AAAI*, pp. 421–425, 1987.

[44] Hansen, E. and Zilberstein, S., "Monitoring and control of anytime algorithms: A dynamic programming approach," *Artificial Intelligence*, vol. 126, no. 1-2, 2001.

[45] Hayes-Roth, B. and Larsson, J. E., "A domain-specific software architecture for a class of intelligent patient monitoring systems.," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 8, no. 2, pp. 149–171, 1996.

[46] Hoang, H., Lee-Urban, S., and Muoz-Avila, H., "Hierarchical plan representations for encoding strategic game AI," *Proc. Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-05)*, 2005.

[47] Horvitz, E., "Principles and applications of continual computation," *Artificial Intelligence*, vol. 126, no. 1-2, 2001.

[48] Horvitz, E., Cooper, G., and Heckerman, D., "Reflection and action under scarce resources: Theoretical principles and empirical study," *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[49] Howard, R., "Information value theory," *IEEE Transactions on Systems Science and Cybernetics*, vol. ssc-2, pp. 22–26, 1966.

[50] Iba, W. and Langley, P., "Unsupervised learning of probabilistic concept hierarchies," *Lecture Notes in Computer Science*, vol. 2049, pp. 39–70, 2001.

[51] Ivanov, Y. A. and Blumberg, B., "Solving weak transduction with EM," *Robotics and Autonomous Systems*, vol. 39, no. 3-4, pp. 129–143, 2002.

[52] Jensen, F. and Liang, J., "drHugin: A system for value of information in bayesian networks," 1994.

[53] Jones, J. and Goel, A., “Hierarchical Judgement Composition: Revisiting the structural credit assignment problem,” in *Proceedings of the AAAI Workshop on Challenges in Game AI, San Jose, CA, USA*, pp. 67–71, 2004.

[54] Jones, J. and Goel, A., “Knowledge organization and structural credit assignment,” in *Proceedings of the IJCAI Workshop on Reasoning, Representation and Learning in Computer Games, Edinburgh, UK*, 2005.

[55] Jones, J. and Goel, A., “Structural credit assignment in hierarchical classification,” in *Proceedings of the International Conference on Artificial Intelligence (ICAI-07)*, (Las Vegas, NV), 2007.

[56] Jones, J. and Goel, A., “Metareasoning for adaptation of classification knowledge,” in *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, (Richland, SC), pp. 1145–1146, International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[57] Jones, J. and Goel, A. K., “Retrospective self-adaptation of an agents domain knowledge: Perceptually-grounded semantics for structural credit assignment,” in *In Proceedings of the AAAI-08 Workshop on Metareasoning*, (Chicago, IL), 2008.

[58] Jones, J. and Goel, A. K., “Metareasoning-based learning for classification hierarchies,” in *SASO-09 Workshop on Meta-Reasoning*, (San Francisco), pp. 123–130, September 2009.

[59] Jones, J., Parnin, C., Sinharoy, A., Rugaber, S., and Goel, A. K., “Teleological metareasoning for automating software adaptation,” in *Third IEEE Conference on Self-Adaptive and Self-Organizing Systems*, (San Francisco), pp. 198–205, September 2009.

[60] Jordan, M. I. and Jacobs, R. A., “Hierarchical mixtures of experts and the EM algorithm,” Tech. Rep. AIM-1440, 1993.

[61] Kaelbling, L. P., Littman, M. L., and Moore, A. P., “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[62] Kohavi, R., “The power of decision tables,” in *Proceedings of the European Conference on Machine Learning*, pp. 174–189, Springer Verlag, 1995.

[63] Laird, J. E., Newell, A., and Rosenbloom, P. S., “SOAR: an architecture for general intelligence,” *Artif. Intell.*, vol. 33, no. 1, pp. 1–64, 1987.

[64] Leake, D. and Ram, A., “Goal-driven learning: Fundamental issues (a symposium report),” *The AI Magazine*, vol. 14, pp. 67–72, 1994.

[65] Leake, D. B., "Experience, introspection and expertise: Learning to refine the case-based reasoning process," *J. Exp. Theor. Artif. Intell.*, vol. 8, no. 3-4, pp. 319–339, 1996.

[66] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[67] Littman, M., Sutton, R., and Singh, S., "Predictive representations of state," 2001.

[68] Madani, O., Lizotte, D. J., and Greiner, R., "The budgeted multi-armed bandit problem.," in *COLT*, pp. 643–645, 2004.

[69] Marthi, B., Russell, S., and Latham, D., "Writing Stratagus-playing agents in concurrent ALisp," in *Proceedings of the IJCAI Workshop on Reasoning, Representation and Learning in Computer Games, Edinburgh, UK*, 2005.

[70] Mataric, M., "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[71] McCallum, A. and Nigam, K., "Employing EM and pool-based active learning for text classification," in *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, (San Francisco, CA, USA), pp. 350–358, Morgan Kaufmann Publishers Inc., 1998.

[72] McCallum, R. A., "Instance-based utile distinctions for reinforcement learning with hidden state," in *Proceedings Twelfth International Conference on Machine Learning*, (San Francisco, CA), pp. 387–396, Morgan Kaufmann, 1995.

[73] Mens, T., Buckley, J., Zenger, M., and Rashid, A., "Towards a taxonomy of software evolution," 2003.

[74] Minsky, M., Singh, P., and Sloman, A., "The St. Thomas common sense symposium: Designing architectures for human-level intelligence," *AI Magazine*, vol. 25, no. 2, pp. 113–124, 2004.

[75] Minsky, M., "Steps towards artificial intelligence," *In E. A. Feigenbaum and J. Feldman eds. Computers and Thought*, pp. 406–450, 1963.

[76] Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y., "Explanation-based learning: A problem solving perspective," in *Machine Learning: Paradigms and Methods* (Carbonell, J., ed.), pp. 63–118, London: MIT Press, 1989.

[77] Mitchell, T., "Generalization as search," *Artificial Intelligence*, vol. 18, pp. 203–226, 1982.

[78] Mitchell, T. M. and Thrun, S. B., "Explanation-based neural network learning for robot control," in *Advances in Neural Information Processing Systems 5, Proceedings of the IEEE Conference in Denver* (Giles, C. L., Hanson, S. J., and Cowan, J. D., eds.), (San Mateo, CA), Morgan Kaufmann, 1993.

[79] Mitchell, T., *Machine Learning.* Boston: McGraw-Hill, 1997.

[80] Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., and Schlimmer, J. C., "Theo: A framework for self-improving systems," in *Architectures for Intelligence* (VanLehn, K., ed.), pp. 323–355, Erlbaum, 1991.

[81] Murdock, B. and Goel, A., "Towards adaptive web agents," in *Proceedings of the Fourteenth IEEE International Conference on Automated Software Engineering*, pp. 335–338, 1999.

[82] Murdock., J. W., *Self-Improvement through Self-Understanding: Model-Based Reflection for Agent Adaptation.* PhD thesis, College of Computing, Georgia Institute of Technology, July 2001.

[83] Murdock, J. W. and Goel, A. K., "Meta-case-based reasoning: Self-improvement through self-understanding," in *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 20, pp. 1–36.

[84] Murdock, J. W. and Goel, A. K., "A functional modeling architecture for reflective agents," in *AAAI-98 workshop on Functional Modeling and Teleological Reasoning*, 1998.

[85] Murdock, J. W. and Goel, A. K., "Learning about constraints by reflection," in *AI '01: Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence*, (London, UK), pp. 131–140, Springer-Verlag, 2001.

[86] Murdock, J. W. and Goel, A. K., "Meta-case-based reasoning: self-improvement through self-understanding," *J. Exp. Theor. Artif. Intell.*, vol. 20, no. 1, pp. 1–36, 2008.

[87] Murdock, W. and Goel, A. K., "Localizing planning using functional process models," in *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS-03)*, 2003.

[88] Pearl, J., *Probabilistic reasoning in intelligent systems: networks of plausible inference.* San Mateo, CA: Morgan Kaufmann, 1988.

[89] Porter, B. W., Bareiss, R., and Holte, R. C., "Concept learning and heuristic classification in weakttheory domains," *Artificial Intelligence*, vol. 45, no. 1-2, pp. 229–263, 1990.

[90] Punch, W., Goel, A. K., and Brown, D. C., "A knowledge-based selection mechanism for strategic control with application in design, assembly and planning," in *International Journal of Artificial Intelligence Tools*, vol. 4, pp. 323–348, 1995.

[91] Raja, A. and Lesser, V. R., "A framework for meta-level control in multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 15, no. 2, pp. 147–196, 2007.

[92] Ram, A. and Leake, D. B., *Goal-Driven Learning.* Cambridge, MA: MIT Press, 1995.

[93] Robertson, P. and Laddaga, R., "Meta-reasoning for multi-spectral satellite image interpretation," in *AAAI-08 Workshop on Metareasoning*, (Chicago, IL), 2008.

[94] Rumelhart, D. E. and Mcclelland, J. L., eds., *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. Volumes 1 & 2. MIT Press, 1986.

[95] Russell, S. and Wefald, E., "Principles of metareasoning," *Artificial Intelligence*, vol. 49, 1991.

[96] Russell, S. and Norvig, P., *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition ed., 2003.

[97] Russell, S. J., "Tree-structured bias," in *AAAI*, pp. 641–645, 1988.

[98] Samuel, A., "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, 1957.

[99] Schmill, M. D., Oates, T., Anderson, M., Joysula, D., Perlis, D., Wilson, S., and Fults, S., "The role of metacognition in robust AI systems," in *AAAI-08 Workshop on Metareasoning*, (Chicago, IL), 2008.

[100] Shapiro, A. D., *Structured induction in expert systems.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987.

[101] Shortliffe, E., *Computer-Based Medical Consultations: MYCIN.* New York: Elsevier/North-Holland, 1976.

[102] Singh, S., James, M. R., and Rudary, M. R., "Predictive state representations: a new theory for modeling dynamical systems," in *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, (Arlington, Virginia, United States), pp. 512–519, AUAI Press, 2004.

[103] Sloman, A., "Varieties of metacognition in natural and artificial systems," in *AAAI-08 Workshop on Metareasoning*, (Chicago, IL), 2008.

[104] Stefik, M., "Planning and meta-planning (MOLGEN: Part 2)," *Artif. Intell.*, vol. 16, no. 2, pp. 141–170, 1981.

[105] Stefik, M., *Introduction to Knowledge Systems.* Morgan Kaufmann, June 1995.

[106] Stroulia, E. and Goel, A. K., "Learning problem solving concepts by reflecting on problem solving," in *Proceedings of the 1994 Euopean Conference on Machine Learning*, 1994.

[107] Stroulia, E. and Goel, A. K., "A model-based approach to blame assignment: Revising the reasoning steps of problem solvers," in *Proceedings of AAAI'96*, pp. 959–965, 1996.

[108] Stroulia, E. and Goel, A., "Functional representation and reasoning in reflective systems," *Journal of Applied Intelligence, Special Issue on Functional Reasoning*, vol. 9, no. 1, pp. 101–124, 1995.

[109] Stroulia, E. and Goel, A. K., "A model-based approach to blame assignment: Revising the reasoning steps of problem solvers," *National Conference on Artificial Intelligence (AAAI-96)*, pp. 959–965, August 1996.

[110] Stroulia, E. and Goel, A. K., "Redesigning a problem-solver's operations to improve solution quality," *15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 562–567, August 1997.

[111] Stroulia, E. and Goel, A. K., "Evaluating problem-solving methods in evolutionary design: the Autognostic experiments," *International Journal of Human-Computer Studies, Special Issue on Evaluation Methodologies*, vol. 51, pp. 825–847, 1999.

[112] Sutton, R., "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[113] Sutton, R. S., Precup, D., and Singh, S. P., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, pp. 181–211, 1999.

[114] Sutton, R. and Barto, A., *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[115] Tadepalli, P. and Russell, S. J., "Learning from examples and membership queries with structured determinations," in *Machine Learning*, vol. 32, pp. 245–295, 1998.

[116] Tishby, N., Pereira, F. C., and Bialek, W., "The information bottleneck method," pp. 368–377, 1999.

[117] Tong, S. and Koller, D., "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, 2002.

[118] Towell, G. G. and Shavlik, J. W., "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1-2, pp. 119–165, 1994.

[119] Tumer, K., Agogino, A., and Wolpert, D., "Learning sequences of actions in collectives of autonomous agents," 2002.

[120] Ulam, P., Goel, A., Jones, J., and Murdock, W., "Model-based guidance for reinforcement learning," *In Proc. IJCAI-05 Workshop on Game Playing*, August 2005.

[121] Ulam, P., Goel, A. K., and Jones, J., "Reflection in action: Model-based self-adaptation in game playing agents," in *AAAI-2004 Workshop on Challenges in Game Playing*, 2004.

[122] Ulam, P., Jones, J., and Goel, A. K., "Using model-based reflection to guide reinforcement learning," in *Fourth AAAI Conference on AI in Interactive Digital Entertainment (AIIDE-08)*, October 2008.

[123] Vapnik, V. and Chervonenkis, A., "On the uniform convergence of relative frequencies of events to their probabilities.," *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.

[124] Viola, P. and Jones, M. J., "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[125] Wasserman, L., *All of Statistics : A Concise Course in Statistical Inference (Springer Texts in Statistics).* Springer, September 2004.

[126] Watkins, C. J., *Learning from delayed rewards.* PhD thesis, Cambridge university, 1989.

[127] Webb, R. H. and Rowe, T. S., "An index of leading indicators for inflation," *Economic Quarterly*, no. Spr, pp. 75–96, 1995.

[128] Weintraub, M. and Bylander, T., "Generating error candidates for assigning blame in a knowledge base," in *International Conference on Machine Learning*, 1990.

[129] Weiss, S. M. and Kulikowski, C. A., *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991.

[130] Whiteson, S., Kohl, N., Miikkulainen, R., and Stone, P., "Evolving keepaway soccer players through task decomposition," *Machine Learning*, vol. 59(1), pp. 5–30, 2005.

[131] Wiewiora, E., "Learning predictive representations from a history," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*, (New York, NY, USA), pp. 964–971, ACM, 2005.

[132] Williams, B. C. and Nayak, P. P., "Immobile robots: AI in the new millennium," *AI Magazine*, vol. 17, no. 3, pp. 16–35, 1996.

[133] Winston, P. H., *Artificial intelligence (3rd ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[134] Wolpert, D. H., "Stacked generalization," Tech. Rep. LA-UR-90-3460, Los Alamos, NM, 1990.

[135] Yun-Long, L. and Ren-Hou, L., "Discovery and learning of models with predictive state representations for dynamical systems without reset," *Knowledge-Based Systems*, January 2009.