Perceptually Grounded Self-Diagnosis and Self-Repair of Domain Knowledge

Joshua K. Jones, Ashok K. Goel

Design & Intelligence Laboratory
School of Interactive Computing
Georgia Institute of Technology

Abstract

We 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, we focus on the self-diagnosis and self-repair of an agent's domain knowledge. The core issue that this article addresses is: what kind of metaknowledge may enable the agent to diagnose faults in its domain knowledge? To address this question, we propose a representation that explicitly encodes metaknowledge in the form of *Empirical Verification Procedures* (EVPs). In the proposed knowledge representation, an EVP may be 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. We present the 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 called Compositional Classification.

Keywords: Knowledge Engineering, Knowledge Representation, Symbol Grounding

---

Email addresses: `jkj@cc.gatech.edu` (Joshua K. Jones), `goel@cc.gatech.edu` (Ashok K. Goel)

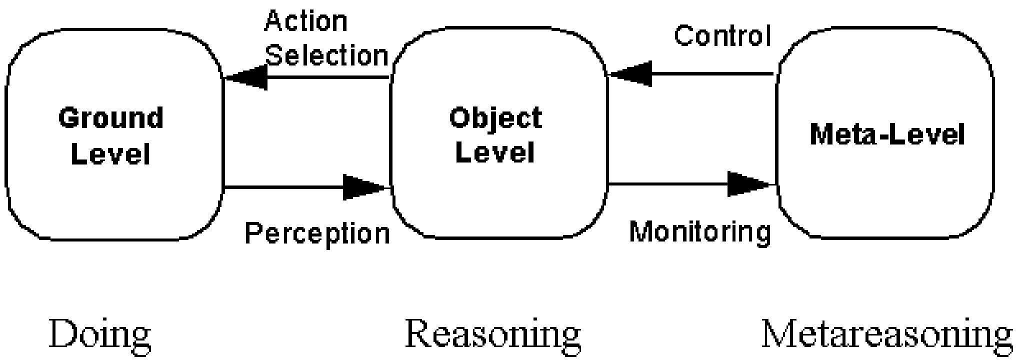


Figure 1: A basic metareasoning architecture, adapted from [7].

1. Introduction

It is generally agreed in AI that the capability of metareasoning, reasoning about reasoning, is essential for achieving human-level intelligence [1] [2] [3] [4]. A canonical metareasoning architecture is depicted in Figure 1. Metareasoning systems extend the basic view of a software agent, where the agent 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 the object level if it becomes apparent that progress is not being made. Cox [5] and Anderson & Oates [6] review AI research on metareasoning.

In this article, we describe work on enabling metareasoning agents, implemented purely in software, to reflect upon and modify the agent's domain knowledge. Note that this research topic is related to, but distinct from, a more common use of metareasoning for agent adaptation – the adaptation of an agent's processing. We are concerned here specifically with adapting declarative domain knowledge and not an agent's process within the context of a domain. 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 *knowledge about domain knowledge (metaknowledge) should be specified in the form of verifiable predictions.*

The next question that arises is: how can one operationalize the verifiable predictions such that the agent can automatically check the correctness of its knowledge? The answer we propose is that each piece of an agent's knowledge may 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). We hypothesize that they are a form of metaknowledge that will enable a system to successfully self-diagnose and repair domain knowledge. The use of explicitly represented EVP knowledge for the diagnosis and repair of domain knowledge is the specific innovation of the research described here. An interesting implication of this hypothesis is that domain knowledge is grounded in perception, because that knowledge will be considered correct only if it leads to accurate predictions about the world and modified to conform to that ideal of correctness otherwise.

In order to test these hypotheses empirically, 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 ([8] [9] [10] [11]), 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., [12] [13]). In fact, this class of problems is so common that it has been identified as a Generic Task [14]. We will call this classification task *Compositional Classification*, and the hierarchy of abstractions an *Abstraction Network*. In particular, we consider the problem of retrospective, failure-driven adaptation of the content of the intermediate abstractions in the Abstraction Network (and *not* its structure) when the classifier makes an incorrect classification.

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 [1] [15] 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 article 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.

In the remainder of this paper, we more formally describe the problem domain to which ANs are applicable (Section 2), formally define EVPs and ANs (Section 3), detail the design of and results obtained from our experiments (Sections 4 & 5), discuss related research (Section 6), and finally conclude (Section 7).

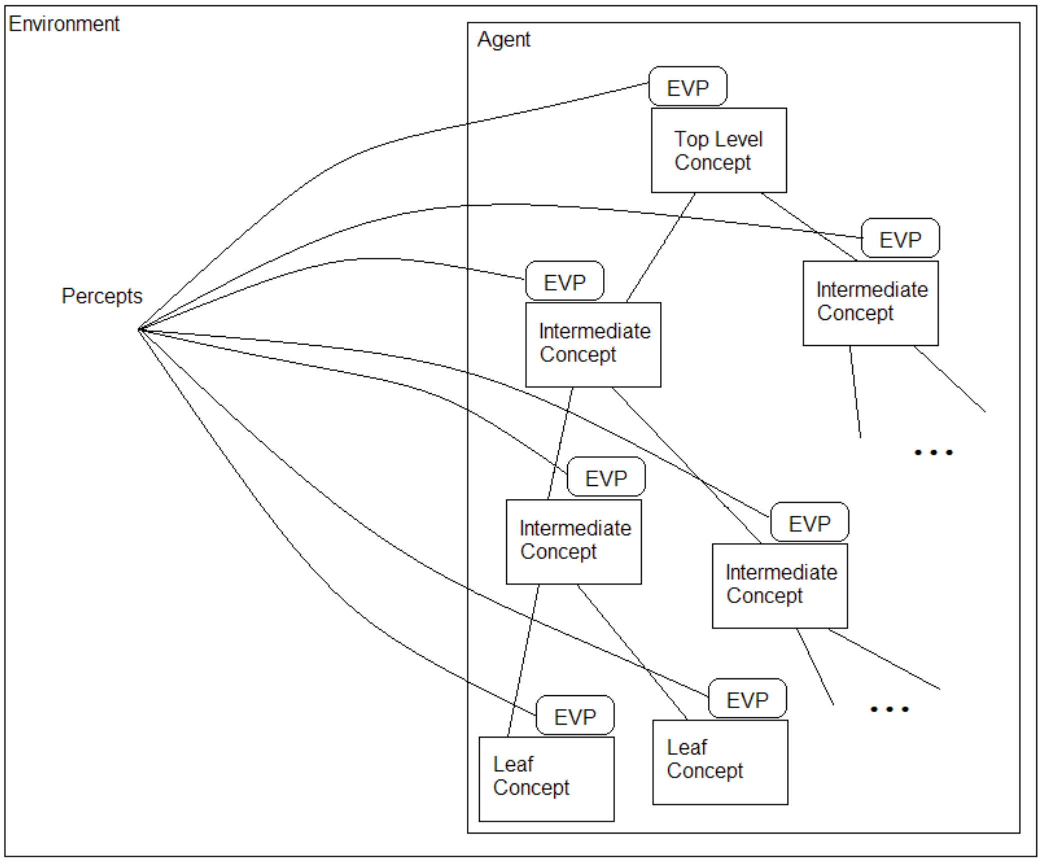


Figure 2: Hierarchical classification knowledge structure with Empirical Verification Procedures grounding concepts in perception.

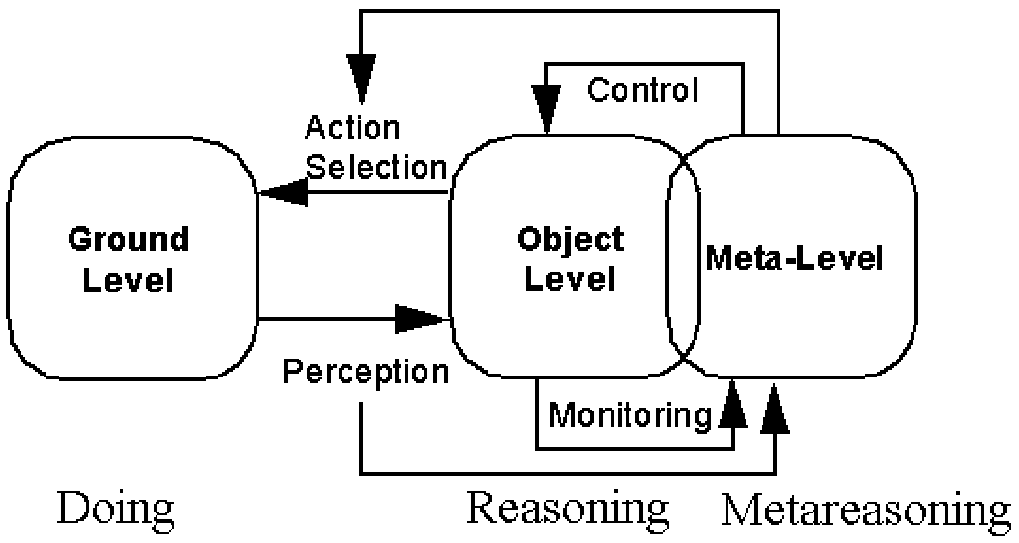


Figure 3: Elaborated metareasoning architecture.

2. Problem Domain

We will consider a general classification problem to require 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.

2.1. Illustrative Example

To make the problem concrete, we will present an example from the turn-based strategy game called FreeCiv (www.freeciv.org). 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.

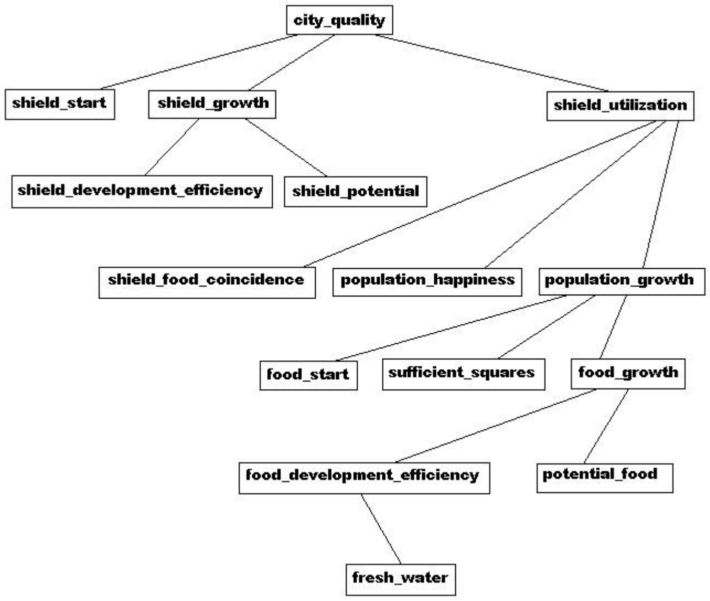


Figure 4: FreeCiv city production estimate classifier.

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 4 illustrates a knowledge hierarchy for this task used by our FreeCiv game-playing agent.

2.2. Compositional Classification

Let t be a discrete random variable representing the class label. Let $S = \{s : s \text{ is empirically determinable and } h[t] > h[t|s]\}$, 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" (defined below). Each member s of S represents a related set of equivalence classes (each value that can be taken by a variable s is a unique classification of the portions of world state abstractly represented by the variable) , 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. 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 article, there is a proper subset of S 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 article, we focus exclusively on problems with this characteristic in order to avoid the need to incorporate strategies for determining information value at this time.

3. Applying Reflection to Compositional Classification Knowledge

In this section, the representational structures used to address the Compositional Classification problem and the reasoning processes that operate over those structures are described formally.

3.1. 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 section. 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.

3.1.1. 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 Section 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 article, 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.

3.2. 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 [16] [12]. A structure used for experimentation in the previously discussed FreeCiv problem is depicted in Figure 4. 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 [12].

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.*

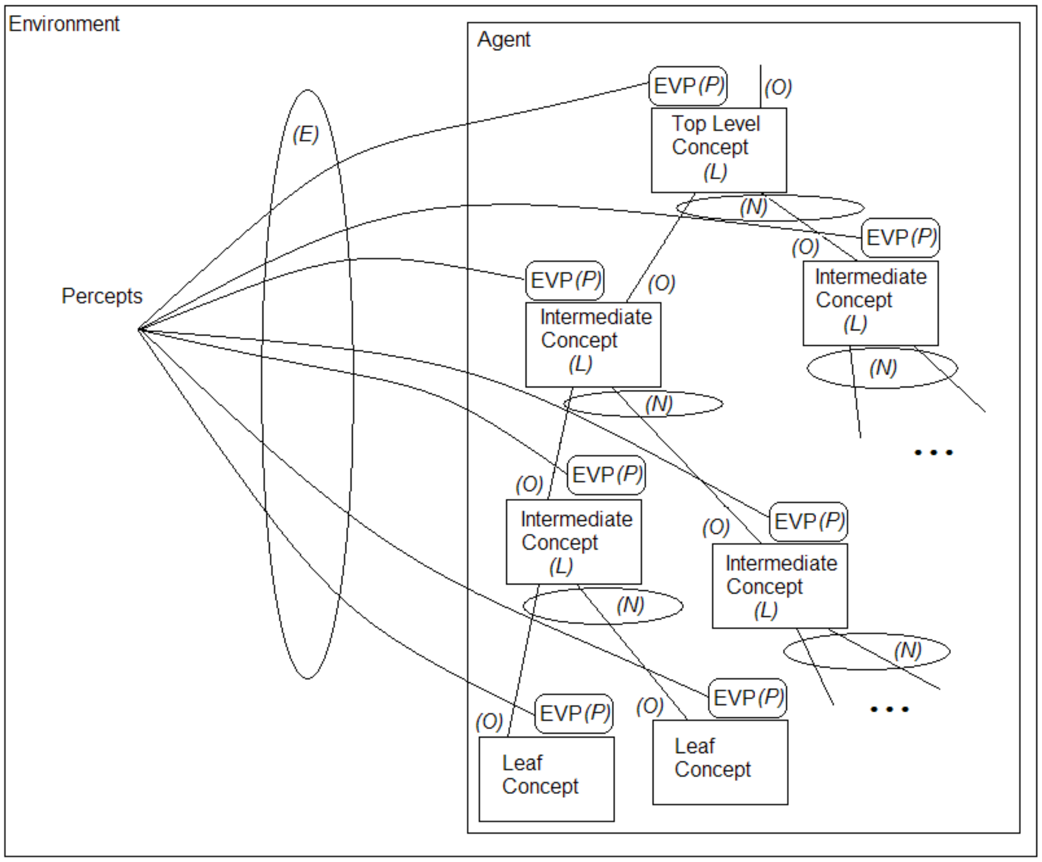


Figure 5: General Abstraction Network architecture with annotations from Definitions 1 and 3. This figure labels the same architecture shown in Figure 2 to illustrate how each of the parts is identified in these formal definitions.

Definition 3. *An* Abstraction Network *(AN) 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 (ANs) to be trees, rather than some more general structure such as directed acyclic graphs (DAGs). 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. Because each learner within an AN is required to conform to the notion of supervised classification learner described in Definition 2, information can be passed between different types of learners – notice that discrete sets of input and output symbols are explicitly required. For learners types such as Artificial Neural Networks (ANNs) that produce real-valued outputs, a wrapper is required to perform discretization and allow the learner to conform to our definition. Figure 5 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 article 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.

3.3. 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

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
```

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.

3.4. Self-Diagnosis and Repair

At some time after classification, the true value of the class label is obtained by the monitoring process (see Figure 3). 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 article, 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.
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 6 (note that *last_value* and *last_input*, used in Table 2, are explained in Section 3.3 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.

Figure 6 depicts an example outcome of the diagnostic procedure described in Table 2. In such a situation, top level feedback indicates a problem with the overall classification produced at the root of the hierarchy. Then, according to the procedure of 2, EVPs at successively deeper levels of the hierarchy are progressively executed, resulting in the examination of various percepts to establish a "frontier" of error-producing nodes, the children of which produced values verified as correct. In the example, 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 identified with children that produced correct values – and thus, for this example, the only node at which the diagnosis procedure can assign blame to incorrect knowledge within a node.

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 3.2 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. In other cases, it may make sense to use Artificial Neural Networks (ANNs) [17] 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

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
```

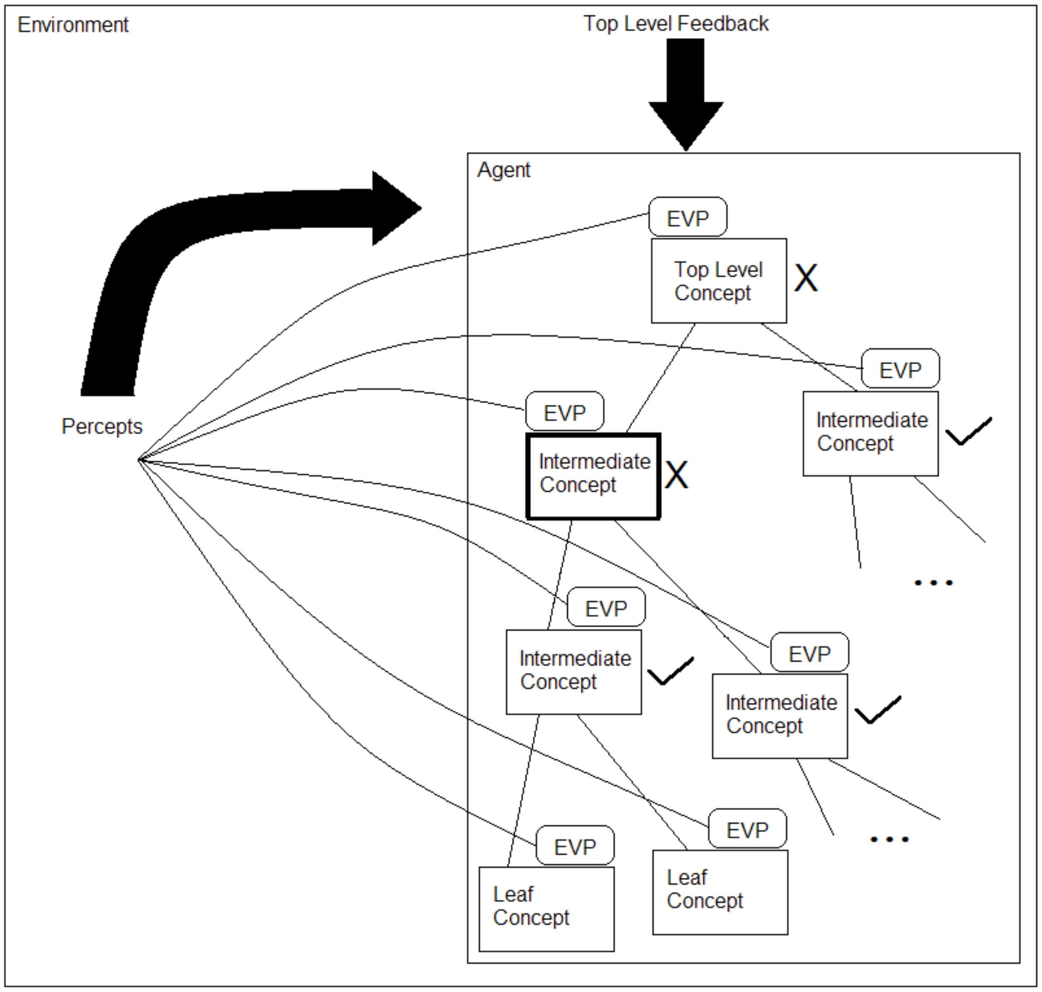


Figure 6: An example outcome of the diagnostic procedure of Table 2.

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.

3.5. Computational Complexity

The effect of hierarchicalization on inference complexity is already well understood and is known to make inference significantly more manageable [12]. However, it is more difficult to characterize the impact of this technique on the computation time required for learning. This is due to wide variation in the conceivable contexts and domains in which AN learning might be deployed. For instance, consider the variety of learner types that may be employed within ANs. If the per-instance learning complexity of the supervised classification learner type used within AN nodes is relatively low for the problem at hand, and scales well with problem dimensionality, the diagnostic overhead of ANs may lead to a higher per-instance learning complexity. However, as the experimental results reported in this paper indicate, ANs generally allow learning from many fewer instances, offsetting the per-instance diagnostic cost. Also, hierarchicalization decreases problem dimensionality for any given internal supervised classification learner, which may have a significant impact on techniques that scale poorly with problem dimensionality. Perhaps the largest variable is the cost of EVP execution within a domain in which ANs are to be employed. EVP details may vary widely, ranging from a simple inspection of the environment to an involved and possibly branching procedure that must be executed (e.g. performing medical laboratory tests). However, in the latter case, collection of data for a standard (non-AN) learning technique is also likely to be very expensive, though these data collection costs are often neglected in analyzing learning complexity. Though space prohibits a complete reproduction of the proof here, we have proven that the diagnostic procedure described above is optimal with respect to maximizing expected decrease in *diagnostic search space* entropy with each EVP execution [18]. Also in [18], an empirical comparison between AN learning and basic Bayes net (BN) learning is presented, showing that AN learning scales favorably with respect to standard BN learning as problem complexity grows. In general, the practitioner should weigh (1) per-instance learning cost of the supervised classification technique to be used, (2) problem dimensionality, and the sensitivity of the learner type to dimensionality, and (3) expected EVP execution cost within the domain in deciding whether ANs are a practical solution in a given problem context.

4. Experimental Design

In this section, 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.

4.1. 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. In this section we define the table-based rote learners with which some of the experiments are performed. The table-based rote learners are an instance of the supervised classification learners of Definition 2.

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 $\forall t \in T, o \in O, (o, t) \rightarrow o$.*

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.

Similar rote learners are discussed by Kohavi in [19]. These learners share the same basic principle as those used in this work – memorize input examples. However, there are some key differences. Most importantly, Kohavi's work described in [19] 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.

5. Experimental Results

In this section, 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) [17] and k-Nearest Neighbor learners (kNNs) [10] working within AN nodes. Beyond experiments that test the central hypothesis of this article, that EVPs provide adequate metaknowledge for an agent to self-diagnose and repair faults in its classification knowledge (Sections 5.1 & 5.2), we also describe experiments dealing with the effects of faulty structural knowledge engineering on AN performance (Section 5.3).

5.1. 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 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 3.4 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 Section 5.1.2. 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 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.

5.1.1. Rote Learners

In this section, we describe results in the synthetic domain using rote learners 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. 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 3.4. 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 7-9 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

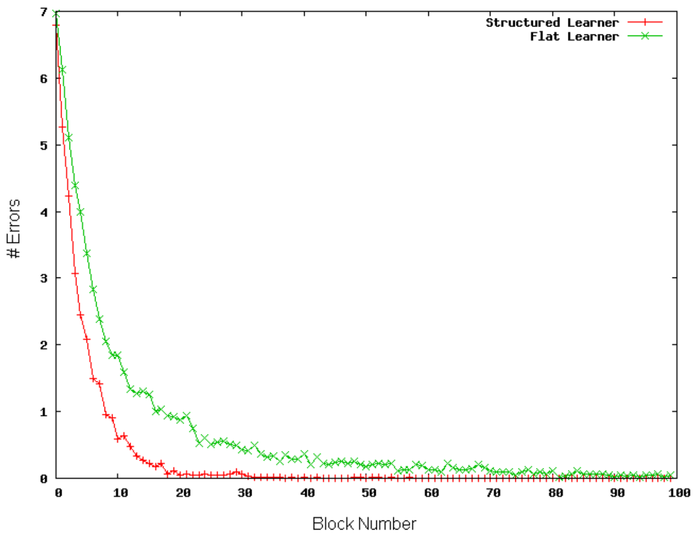


Figure 7: Per-block error rates of AN-rote learners vs. flat rote learners for layer sizes 4, 2, 1.

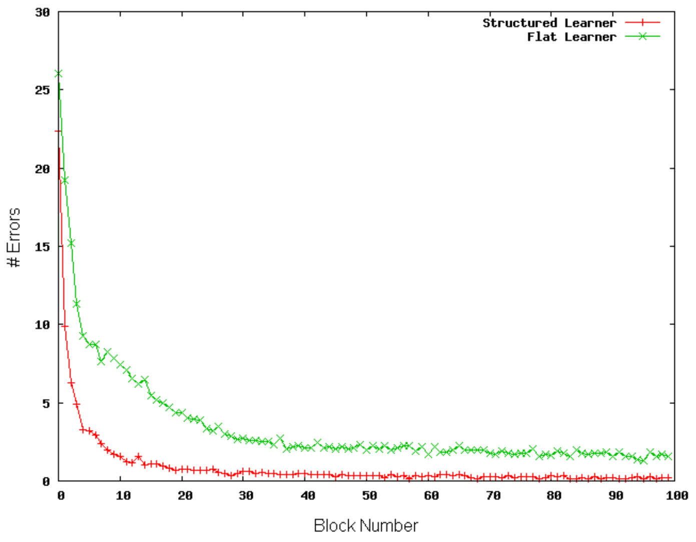


Figure 8: Per-block error rates of AN-rote learners vs. flat rote learners for Layer sizes 8, 4, 2, 1.

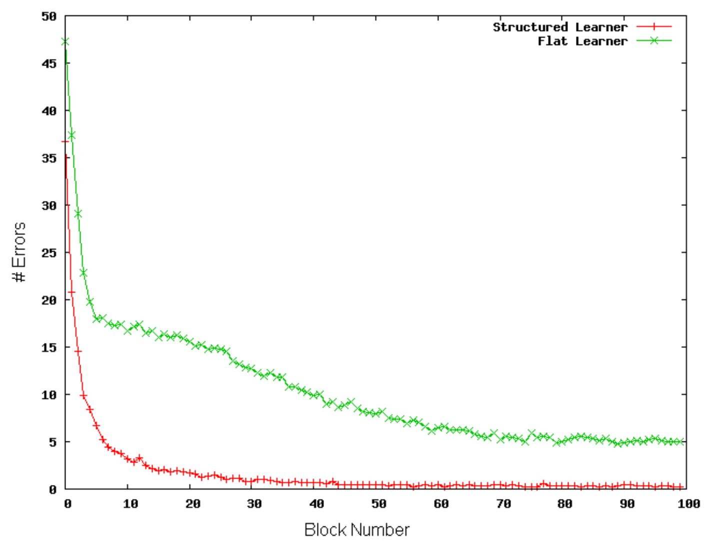


Figure 9: Per-block error rates of AN-rote learners vs. flat rote learners for Layer sizes 16, 8, 4, 2, 1.

be seen in these results. This benefit is due to the restriction bias imposed on the hypothesis space available to the learner by the hierarchical knowledge structure and the semantic constraints encoded by EVPs.

5.1.2. Artificial Neural Networks

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 networks (ANNs). 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 3.4, 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 10-12. 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 12, the training set contains 1,000 examples, while the test set contains 10,000 examples. For the experiments shown in Figures 10 and 11, both the training and test sets contained 1,000 examples. We wished to ensure that we were sampling the problem space sufficiently to achieve consistent results, and once we found that this could be done with a test set consisting of 1,000 examples we continued to test with this smaller, more rapidly completing test set size. In each case, we ran the complete experiment 5 times (re-randomizing all learners and the fixed AN each time, etc.), and Figures 10-12 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.

5.1.3. 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 13-15,

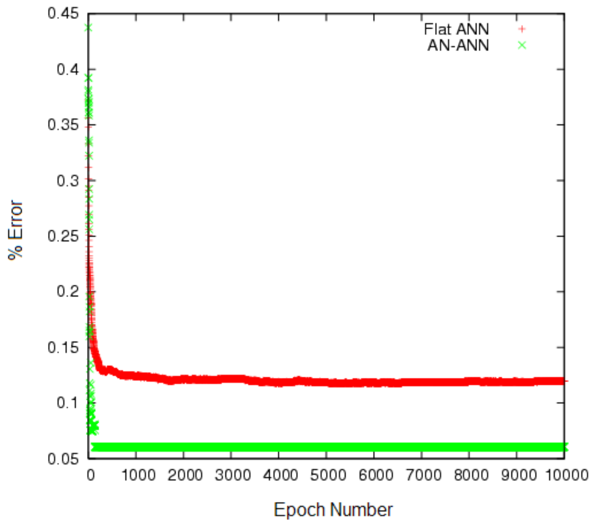


Figure 10: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 8-4-2-1, 3 choices per node.

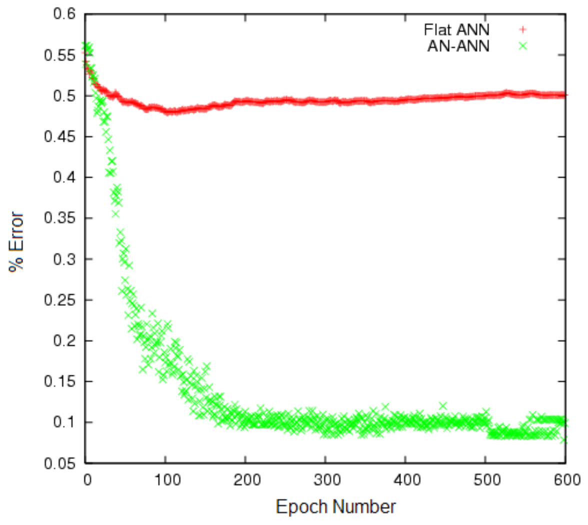


Figure 11: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 8-4-2-1, 4 choices per node.

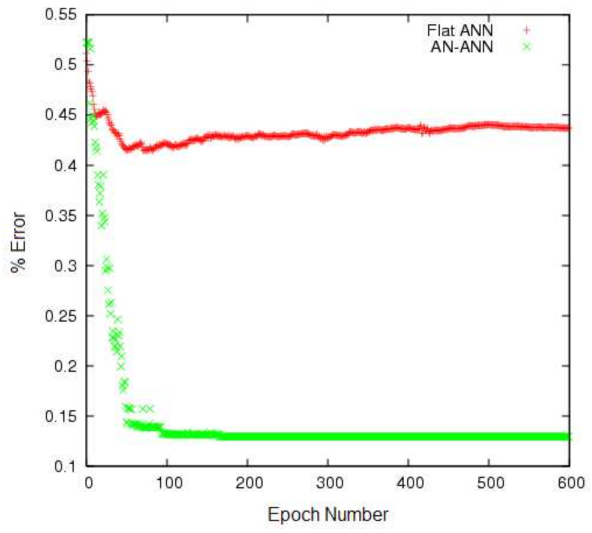


Figure 12: % Per-epoch error rates (% error) of AN-ANN vs. unaugmented ANNs for layer sizes 16-8-4-2-1, 3 choices per node.

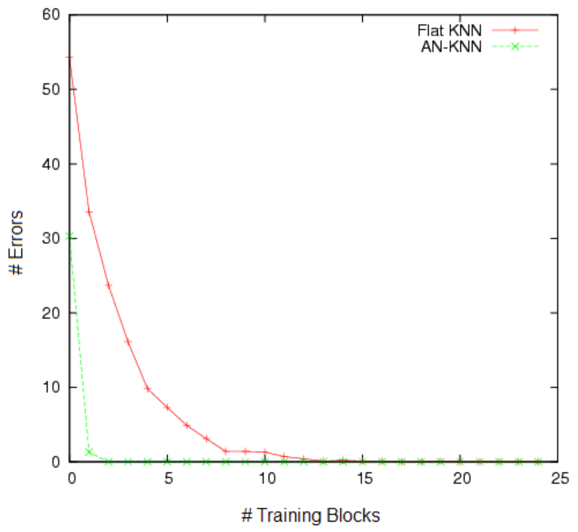


Figure 13: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 4-2-1, 4 choices per node.

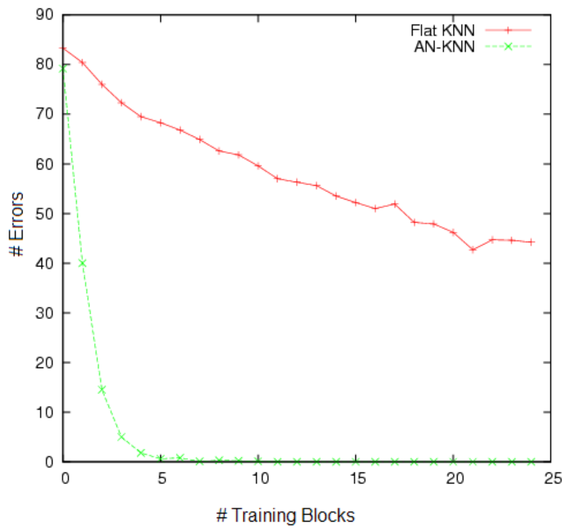


Figure 14: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 4-2-1, 8 choices per node.

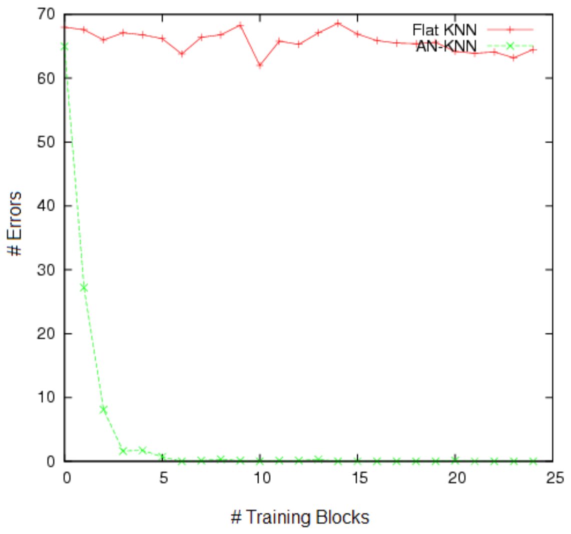


Figure 15: Per-block error rates of AN-kNN vs. unaugmented kNNs for layer sizes 16-8-4-2-1, 4 choices per node.

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.

5.2. Real Domains

5.2.1. FreeCiv City Location

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

For the FreeCiv city resource production prediction task described at the beginning of Section 2, we use the Compositional Classifier depicted in Figure 4, producing predictive classifications of map locations in a sequence of games. Within each node, we use a simple rote learner, defined in Section 4.1, with a threshold of 5. We have experimentally compared an AN-based learner using the network depicted in Figure 4 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. As mentioned previously, the effect of hierarchicalization on inference complexity is already well understood and is known to make inference significantly more manageable [12]. 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 4 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 3.1, 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, 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 on-line 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 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.

The results of the experiment are summarized in Table 3 and are shown in

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%

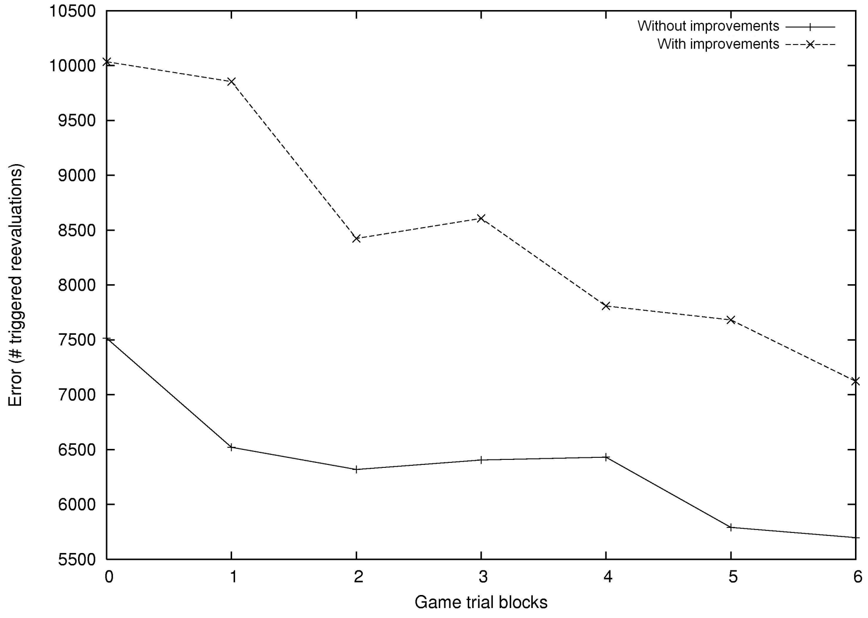


Figure 16: Average error rates by block in each FreeCiv trial.

detail for the AN learners across each block of games in Figure 16. 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 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.

5.2.2. 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 17) 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 17 based on some studies of economic indicators [20][21][22]. 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 17 can be observed before classification, while the remainder are future values at classification time, available only in retrospect.

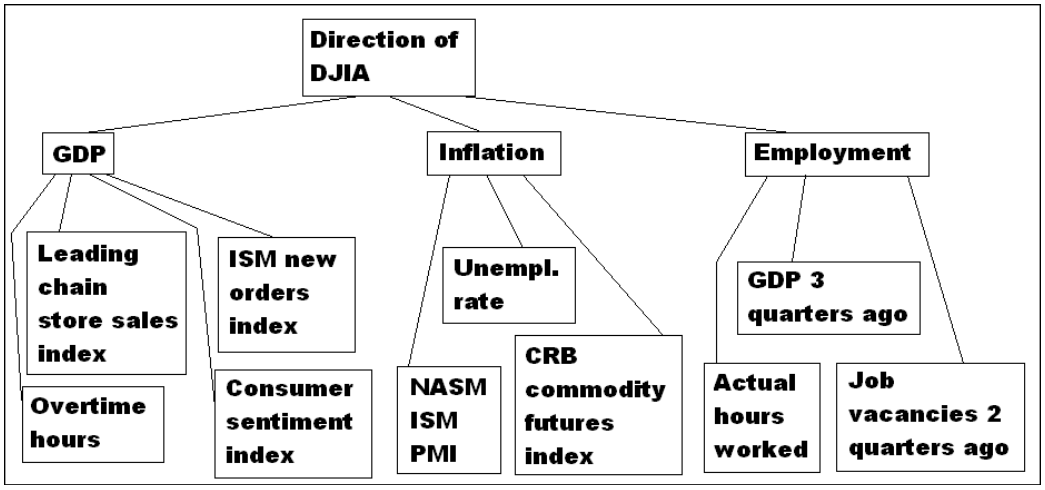


Figure 17: DJIA Abstraction Network

We used data from Jan 1960 - Nov 2005, yielding a total of 497 training examples. 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 18, 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.

5.2.3. 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 19 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

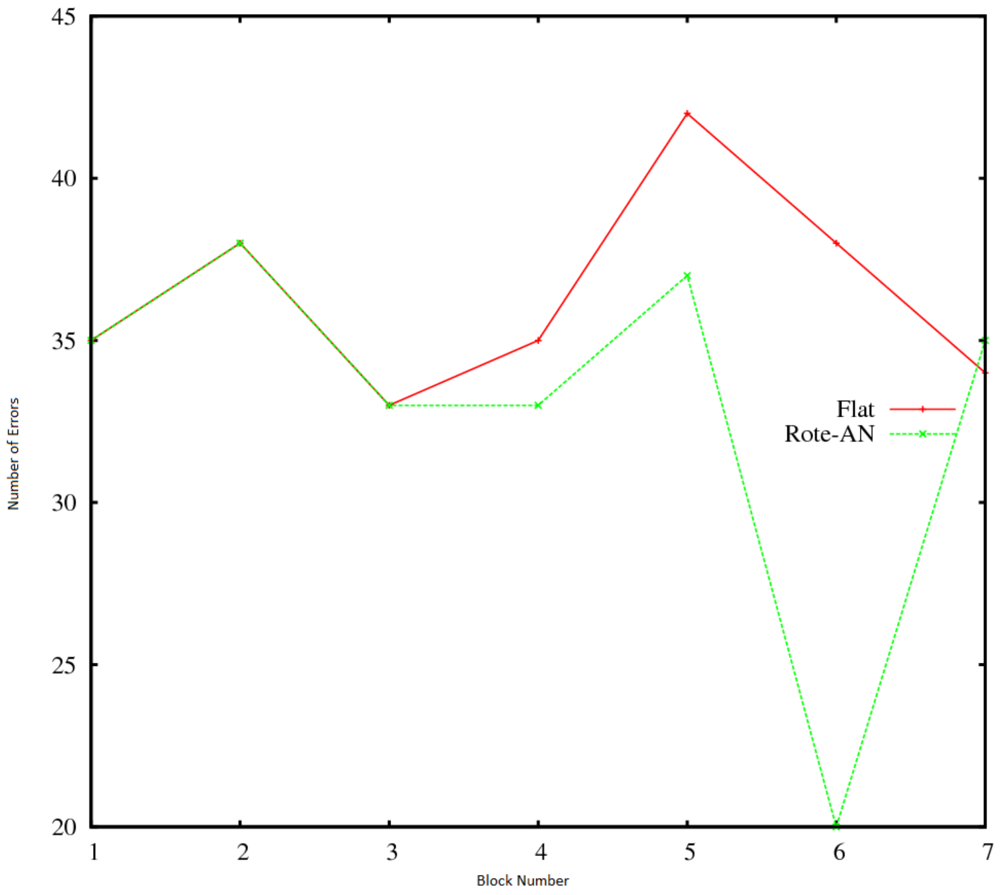


Figure 18: Average error rates by block in each DJIA trial.

data but with randomized initialization of the learners' knowledge.

Results for these experiments are shown in Figure 20. 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 enough,

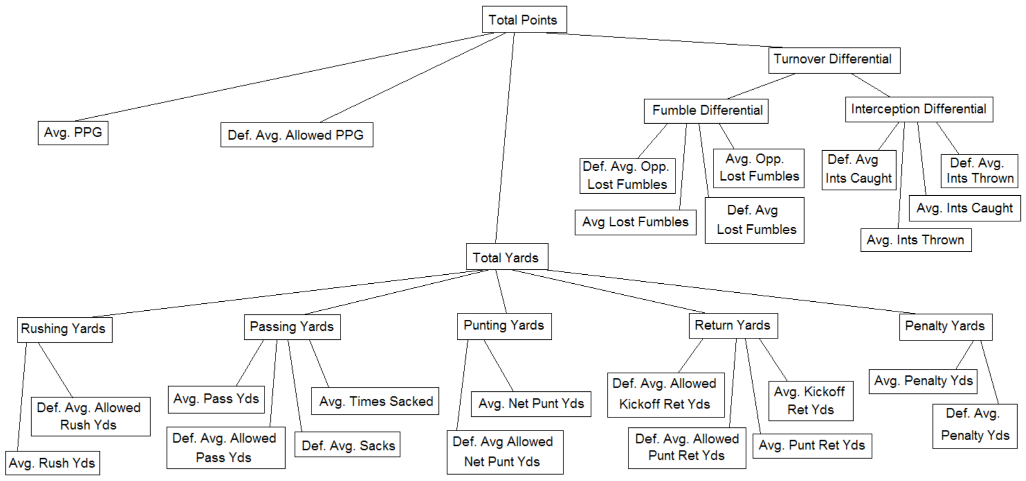


Figure 19: AN structure used in NFL prediction problem.

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.

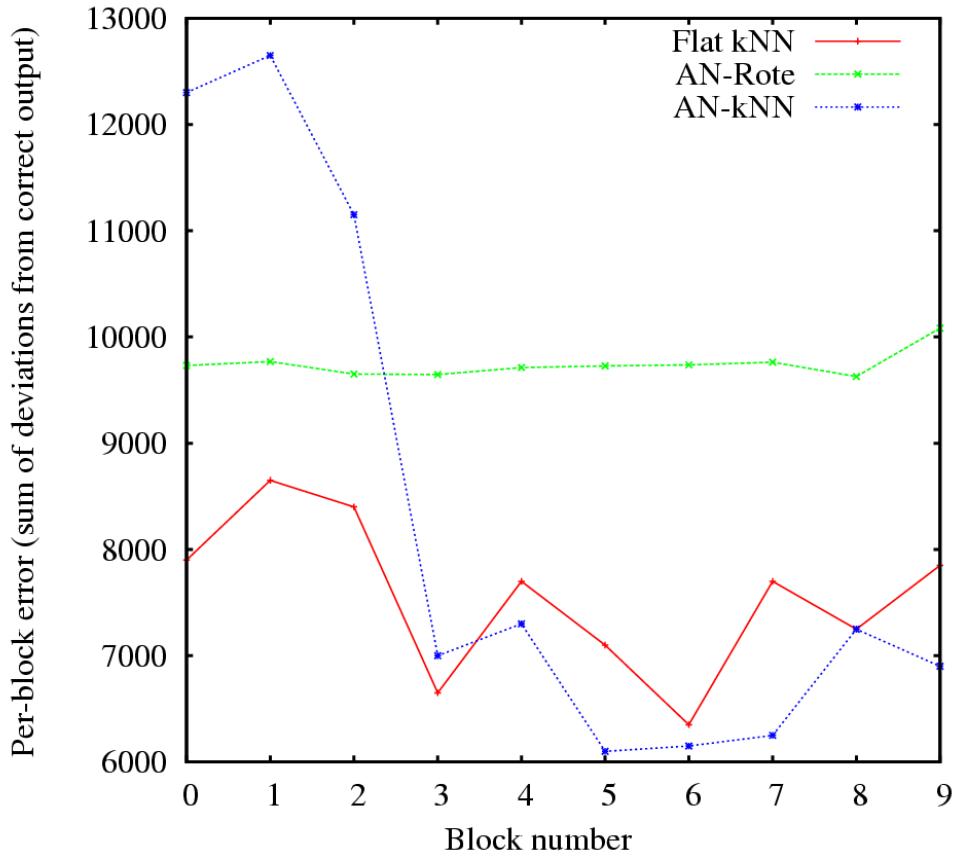


Figure 20: Per-block error vs. block number for various learner types in the NFL football domain.

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.

5.3. Effects of Degraded Knowledge Engineering

We have performed two sets of experiments in the synthetic domain of Section 5.1 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 21. 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.

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 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

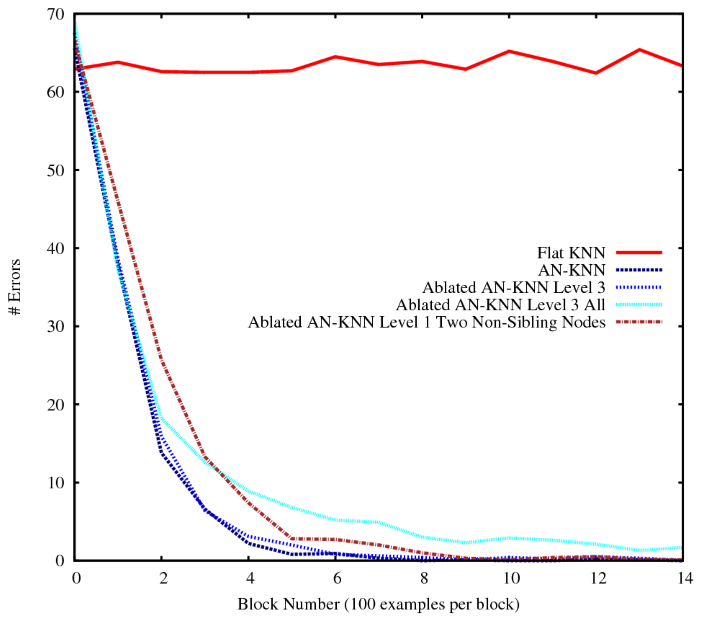


Figure 21: Results of ablating (groups of) individual nodes from an AN learner.

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 22. 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

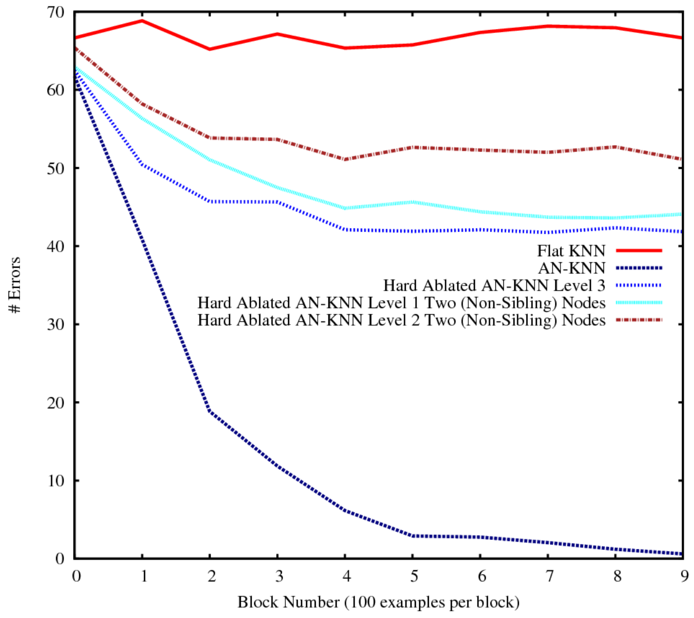


Figure 22: Results of ablating (groups of) subtrees from an AN learner.

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 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 article 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.

5.4. Additional Comments

One would naturally also wish to determine whether, beyond the hierarchical knowledge representation integral to ANs, there is also generalization power imparted particularly 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. However, the experiments reported on in this paper do not serve to distinguish the inductive bias provided by EVPs from the inductive bias imparted by the hierarchical structuring of knowledge. We have performed an extensive set of experiments that compare learning hierarchically structured knowledge both with and without EVPs. Space prohibits a complete discussion of those experiements here, but such a discussion can be found in Jones [18]. The general finding is that while, as expected, the hierarchical structuring of knowledge does itself provide some benefit in terms of generalization power, EVPs also provide additional generalization power. That is, of three tested learner types, flat learners fare the worst, hierarchical learners without EVPs perform somewhat better, and the best performance is realized by hierarchical learners that do make use of EVPs. Further, when hierarchical learners have EVPs at some, but not all nodes, there is a gradual degredation of performance as fewer and fewer EVPs are made avaliable. Thus, it is worth adding EVPs at some nodes in a hierarchical learner even if it is not possible to do so at all nodes.

6. Related Research

Beyond the classifiers which we have integrated directly with the Augur system (kNNs [10] and ANNs [17]), 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.

6.1. Learning With Structured Representations

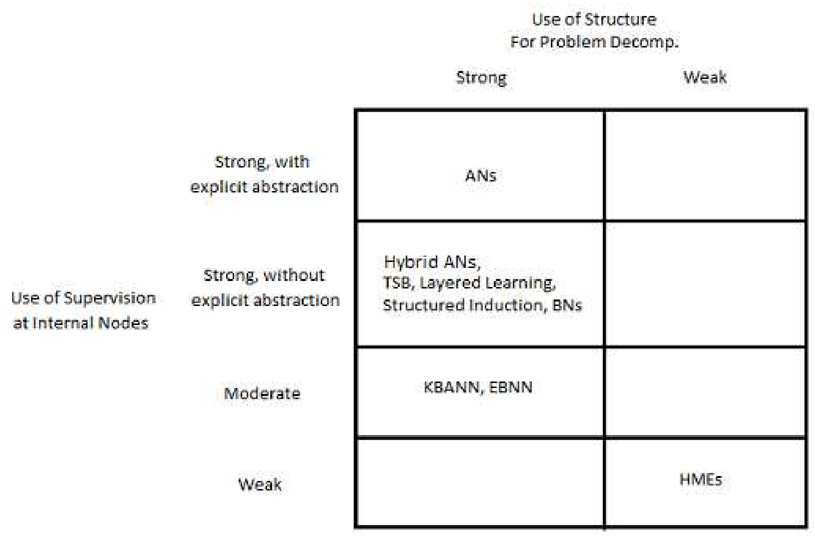
There is a myriad of work on learning that makes use of structured representations. In this section, we highlight and discuss some research that is particularly pertinent to the techniques we have developed in this work. For the purposes of comparison, we will situate each technique discussed in this 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 fit well within the taxonomy suggested by these axes, and the following subsection covers those that do not fit as neatly.

6.1.1. 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 4 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) [13][23] 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

Table 4: 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

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 [23] 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 [23], 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 [24] 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 [25] 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 article 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 [26] and Explanation-based NNs [27] 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 [28] 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) [29] 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.

6.2. 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 [1]. Samuel [15] first identified the problem in his work on checkers playing programs. Minsky [1] identified credit assignment as one of the core problems in AI.

AI systems have used a variety of techniques for self-diagnosis and self-repair in different kinds of agents addressing different tasks in different domains. For example, while Cox & Ram [30] describe the use of explanation patterns for self-diagnosis in a planning agent, Stroulia & Goel [31] describe the use of model-based self-diagnosis and self-repair in a situated, reactive agent. In the model-based method, a self-model that explicitly represents the agent's knowledge and processing 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. Birnbaum et al. [32] describe model-based adaptation in planning agents. Fox & Leake [33] [34] demonstrate how model-based introspective reasoning can help refine case indexing in a case-based planning agent and lead to better case retrieval. For instance, Williams' work on immobots [35] imbues systems viewed as immobile robots with the ability to self-regulate and self-repair by making physical configuration changes when problems are detected.

The REM [36] [37] [38] and Autognostic [39] [40] [31] 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. Jones et al. [41] describe GAIA, a system that uses REM's technique for model-based self-adaptation in interactive game-playing agents. Goel & Jones [42] related work on REM with the research described here.

Ladagga [43] and Robertson & Ladagga [44] have also applied the AI techniques of model-based self-adaptation to software agents. Williams & Mayak's work on immobots [35] imbues systems viewed as immobile robots with the ability to self-regulate and self-repair by making physical configuration changes when problems are detected. We believe that an especially productive research direction is the use of model-based self-adaptation for steering [45] or localizing [46] reinforcement learning. Anderson et al. [47] provide a review of some of these model-based techniques for designing autonomous systems.

6.3. Intelligent Agents and Knowledge Systems

Work in the knowledge-based systems community on expert systems has advocated the use of empirical methods to ensure the correctness of the domain knowledge encoded in a system [48] [49]. These empirical methods involve using a system to solve a predefined set of problems constituting a set of test cases with known desired solutions. EVPs are related to these methods in that they can be viewed as implicitly defining a *universal* set of test cases for each concept within a hierarchy of domain knowledge. That is, by defining the semantics of a concept in terms of actions and perceptions within the environment, EVPs allow the appropriate value of a concept to be determined (for a cost) in any circumstance. Further, in this work we advocate the enforcement of desired semantics at both a holistic level (due to the incremental diagnostic method) and at the more fine-grained level of individual concepts. This kind of semantic enforcement facilitates the automatic repair of faulty knowledge.

In this work, we give an account of a diagnostic method useful for repair and refinement of attribute-based concept classification knowledge within a known, fixed concept hierarchy. We do not give an account of how either the concepts themselves (i.e. the semantic definition of concepts via EVPs), or the structural arrangement of those concepts into a useful hierarchy might be automatically derived. Rather, we expect knowledge engineers to manually encode these aspects of domain knowledge. However, there has been much work on learning of classification knowledge in general [50] [51], in various settings. Work on Tree-Structured Bias [13], described above, gives one account of how the hierarchical knowledge might be discovered from a set of background facts. Fisher [52] gives another account of incremental conceptual clustering.

Another approach to the automatic generation of concepts and relevant relationships from background facts and examples is Inductive Logic Programming (ILP) [53] [54]. However, the representational form of knowledge manipulated by ILP is substantially different from that encoded in an Abstraction Network. Thus, the problem framing demanded for the application of ANs is distinct from that demanded for the application of ILP. Practically speaking, it is likely that the form in which background knowledge is available (or can be produced) in a given problem setting will determine the relative applicability of ANs vs ILP-based methods.

7. Conclusion

The central claim that we make in this article 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 Section 5 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. We do experiment in some non-synthetic domains, with positive results. While ANs have not yet been deployed in the field, we feel that the domains tested, particularly the economic and game-playing settings, are good candidates for practical use of the technology. In particular, ANs have the potential to be valuable tools for validating models of economic and other complex systems. In such settings, both successful learning and problems with concept learning (i.e. vacillation within nodes) can provide valuable feedback about the quality of the structure and conceptual content of the model.

The view taken here is that knowledge has meaning because it entails predictions about perceptions, and that conceptual knowledge is valuable in the extent to which it ultimately contributes to action selection. This perspective on the 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 3.4.

We have also presented empirical results of the impact of degraded knowledge engineering on the effectiveness of learning using an Abstraction Network (Section 5.3). 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.

Acknowledgements

We would like to acknowledge Tom Dietterich and Charles Isbell for their helpful comments on earlier versions of this work. We also thank Tom

Mitchell for providing code for training Artificial Neural Networks that served as the basis of the ANN code used in our experiments. This research is supported by an NSF (SoD) Grant (#0613744) on Teleological Reasoning in Adaptive Software Design.

References

[1] M. Minsky, Steps towards artificial intelligence, In E. A. Feigenbaum and J. Feldman eds. Computers and Thought (1963) 406–450.

[2] M. Minsky, P. Singh, A. Sloman, The St. Thomas common sense symposium: Designing architectures for human-level intelligence, AI Magazine 25 (2004) 113–124.

[3] R. Brachman, Systems that know what they are doing, IEEE Intelligent Systems (2002) 67–71.

[4] M. Cox, A. Raja (Eds.), Metareasoning: Thinking about Thinking, MIT Press, 2011.

[5] M. T. Cox, Metacognition in computation: A selected research review, Artif. Intell. 169 (2005) 104–141.

[6] M. L. Anderson, T. Oates, A review of recent research in metareasoning and metalearning, AI Magazine 28 (2007) 7–16.

[7] M. T. Cox, A. Raja, Metareasoning: A Manifesto, Technical Report, BBN TM-2028, BBN Technologies (2007).

[8] A. K. Goel, N. Soundararajan, B. Chandrasekaran, Complexity in classificatory reasoning, in: AAAI, pp. 421–425.

[9] B. Chandrasekaran, A. Goel, From numbers to symbols to knowledge structures: Artificial Intelligence perspectives on the classification task, IEEE Trans. Systems, Man & Cybernetics 18 (1988) 415–424.

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

[11] W. J. Clancey, Heuristic classification., Artificial Intelligence 27 (1985) 289–350.

[12] A. Goel, T. Bylander, Computational feasibility of structured matching, IEEE Trans. Pattern Anal. Mach. Intell. 11 (1989) 1312–1316.

[13] S. J. Russell, Tree-structured bias, in: AAAI, pp. 641–645.

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

[15] A. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 3 (1957) 210–229.

[16] T. Bylander, T. Johnson, A. Goel, Structured matching: a task-specific technique for making decisions, Knowledge Acquisition 3 (1991) 1–20.

[17] D. E. Rumelhart, J. L. Mcclelland (Eds.), Parallel distributed processing: Explorations in the microstructure of cognition, volume Volumes 1 & 2, MIT Press, 1986.

[18] J. Jones, Empirically-Based Self-Diagnosis and Repair of Domain Knowledge, Ph.D. thesis, Georgia Institute of Technology, 2010.

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

[20] Leading indicators of employment, Australian Economic Indicators 1350.0 (2004).

[21] R. H. Webb, T. S. Rowe, An index of leading indicators for inflation, Economic Quarterly (1995) 75–96.

[22] RBC US leading economic indicator, Economics Department, RBC Financial Group (2005).

[23] P. Tadepalli, S. J. Russell, Learning from examples and membership queries with structured determinations, in: Machine Learning, volume 32, pp. 245–295.

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

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

[26] G. G. Towell, J. W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1994) 119–165.

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

[28] M. I. Jordan, R. A. Jacobs, Hierarchical Mixtures of Experts and the EM Algorithm, Technical Report AIM-1440, 1993.

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

[30] M. Cox, A. Ram, Introspective multistrategy learning: On the construction of learning strategies, Artificial Intelligence 112 (1999) 1–55.

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

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

[33] S. Fox, D. B. Leake, Using introspective reasoning to refine indexing, in: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.

[34] S. Fox, D. Leake, Introspective reasoning for index refinement in case-based reasoning, Journal of Experimental and Theoretical Artificial Intelligence 13 (2001) 63–88.

[35] B. C. Williams, P. P. Nayak, Immobile robots: AI in the new millennium, AI Magazine 17 (1996) 16–35.

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

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

[38] J. W. Murdock, A. K. Goel, Meta-case-based reasoning: self-improvement through self-understanding, J. Exp. Theor. Artif. Intell. 20 (2008) 1–36.

[39] E. Stroulia, A. Goel, Functional representation and reasoning in reflective systems, Journal of Applied Intelligence, Special Issue on Functional Reasoning 9 (1995) 101–124.

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

[41] J. Jones, C. Parnin, A. Sinharoy, S. Rugaber, A. K. Goel, Teleological metareasoning for automating software adaptation, in: Third IEEE Conference on Self-Adaptive and Self-Organizing Systems, San Francisco, pp. 198–205.

[42] A. Goel, J. Jones, Metareasoning for self-adaptation in intelligent agents, in: M. Cox, A. Raja (Eds.), Metareasoning: Thinking about Thinking, MIT Press, 2011.

[43] R. Laddaga, Guest editor's introduction: Creating robust software through self-adaptation, IEEE Intelligent Systems 14 (1999) 26–29.

[44] P. Robertson, R. Laddaga, Model Based Diagnosis and Contexts in Self Adaptive Software, 2005, pp. 112–127.

[45] M. L. Anderson, T. Oates, W. Chong, D. Perlis, The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance, J. Exp. Theor. Artif. Intell. 18 (2006) 387–411.

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

[47] M. L. Anderson, S. Fults, D. P. Josyula, T. Oates, D. Perlis, S. Wilson, D. Wright, A self-help guide for autonomous systems, AI Magazine 29 (2008) 67–73.

[48] S. Hanks, M. E. Pollack, P. R. Cohen, Benchmarks, test beds, controlled experimentation, and the design of agent architectures, AI Mag. 14 (1993) 17–42.

[49] J. Baumeister, Advanced empirical testing, Know.-Based Syst. 24 (2011) 83–94.

[50] P. Langley, Elements of machine learning, Morgan Kaufmann series in machine learning, Morgan Kaufmann, 1996.

[51] T. M. Mitchell, Machine Learning, McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[52] D. H. Fisher, Knowledge acquisition via incremental conceptual clustering, Machine Learning (1987).

[53] S. Muggleton, Inductive Logic Programming: Theory and methods, The Journal of Logic Programming 19-20 (1994) 629–679.

[54] Y. Kavurucu, P. Senkul, I. H. Toroslu, Concept discovery on relational databases: New techniques for search space pruning and rule quality improvement, Know.-Based Syst. 23 (2010) 743–756.