Metareasoning-based Learning for Classification Hierarchies

Joshua Jones & Ashok K. Goel
Design Intelligence Laboratory, School of Interactive Computing
Georgia Institute of Technology, Atlanta, USA 30332
{jkj, goel}@cc.gatech.edu

***Abstract*—This paper takes a metareasoning-based approach to classification learning, framing the learning problem as one of self-diagnosis and self-adaptation. Artificial Intelligence (AI) research on metareasoning for agent self-adaptation has generally focused on modifying the agent's reasoning processes. In this paper, we describe the use of metareasoning for retrospective adaptation of the agent's domain knowledge. In particular, we consider the use of meta-knowledge for structural credit assignment in a classification hierarchy when the classifier makes an incorrect prediction. We present a scheme in which the semantics of the intermediate abstractions in the classification hierarchy are grounded in percepts in the world, and show that this scheme enables self-diagnosis and self-repair of knowledge contents at intermediate nodes in the hierarchy. We also provide an empirical evaluation of the technique.**

I. Introduction

It is generally agreed in AI that the capability of metareasoning is essential for achieving human-level intelligence [1] [2]. One of the many uses of metareasoning is self-adaptation in an intelligent agent. It is useful to make a few distinctions here. First, adaptations in an agent can be retrospective (i.e., when the agent fails to achieve a goal in its given environment [3] [4] [5] [6]), or proactive (i.e., when the agent is asked to operate in a new task environment [7] [8]). Secondly, adaptations can be either to the deliberative element in the agent architecture [3] [4] [5] [8], or the reactive element [9], or both. Thirdly, adaptations to the deliberative element may be modifications to its reasoning process (i.e., to its task structure, selection of methods, or control of reasoning [3] [4] [8]), or to its domain knowledge (i.e., the content, representation and organization of its knowledge [5]), or both.

A core problem in self-adaptation is that of credit (or blame) assignment [10] [11]. It is useful to distinguish between two kinds of credit assignment problems: temporal and structural. In temporal credit assignment, given a sequence of many actions by an agent that leads to a failure, the task is to identify the actions(s) responsible for the failure. In structural credit assignment (SCA), given an agent composed of many knowledge and reasoning elements that fails to achieve a goal, the task is to identify the element(s) responsible for the failure. Metareasoning for self-adaptation typically addresses the latter problem of SCA. It is useful to note the close relationship between agent self-adaptation and agent learning: the use of metareasoning for self-adaptation views learning as a deliberative, knowledge-based process of self-diagnosis and self-repair.

In this paper, we describe work on using metareasoning for repairing an agent's 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]). We will call this classification task *compositional classification*, and the hierarchy of abstractions an *abstraction network*.

In particular, we consider the problem of retrospective adaptation of the content of the intermediate abstractions in the abstraction network (and *not* its structure) when the classifier makes an incorrect classification. Note that once SCA becomes a core problem: given the error at the root node, the problem now is to identify the intermediate abstractions in the abstraction network responsible for the error. In this paper we explore the following hypothesis: if the semantics of the domain concepts that form the intermediate abstractions in a classification hierarchy can be grounded in predictions about percepts in the world, then meta-knowledge in the form of verification procedures associated with those domain concepts enables a metareasoner to address the structural credit assignment problem. In the case of compositional classification, this means that intermediate abstractions in the abstraction network are chosen such that each abstraction corresponds to a prediction about percepts in the world, meta-knowledge comes in the form of verification procedures associated with the abstractions, and metareasoning invokes the appropriate verification procedures to perform structural credit assignment and then adapt the abstractions. The verification procedures explicitly encode the grounding of intermediate abstractions in percepts from the environment.

II. Compositional Classification

Let T be a discrete random variable representing the class label. Let $S = \{s : s$ is empirically determinable and $h[T] > h[T|s]\}$, where $h[x]$ denotes the entropy of x. S is a set of discrete random variables that have nonzero mutual information with the class label and are *empirically determinable*, meaning that there is some way to interact with the environment to determine which value has been taken by each member of S. It is crucial to note that in general this interaction will not be possible until some time after the classification has been produced. That is, we expect that more information will be available during diagnosis and learning, when the system is retrospectively reflecting on a past mistake, than was available at the time that the classification was made. Each member s of S represents a related set of equivalence classes, where each value taken by s is a unique equivalence class. A problem instance is generated by jointly sampling the variables in $S \cup T$.

Empirical determinability captures the notion of perceptual grounding of concepts, 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 problem instance is directly observable at some later time after classification has occurred. In general, some experiment may need to be performed in order to observe the value of some $s \in S$. By performing the necessary experiments, we can obtain the true values of intermediate nodes in order to perform self-diagnosis over the knowledge structure used for the classification. We call the problem of predicting T in such a setting *compositional classification*. In order to make such predictions, our agent will make use of a structured knowledge representation called an *abstraction network*, defined in the next section. This representation will capture knowledge about the relationships between variables in S. Knowledge repair will be required if the distributions $\mathcal{P}(s|K), s \in S \cup T, K \subseteq S$ are not always accurately known by the agent, but must instead be inferred from experience.

III. Abstraction Networks

A. Representation

Here we formally define the knowledge representation used at the object level for the compositional classification task. This representation is annotated with meta-knowledge used by meta-level reasoning process for self-diagnosis. We call this diagnostic self-knowledge *empirical verification procedures*, described in more detail below.

The knowledge structure contains a node for each $s \in S \cup T$. These nodes are connected in a hierarchy reflecting direct dependence relationships organized according to background knowledge. Each node will handle the subproblem of predicting the value of the variable with which it is associated given the values of its children.

Definition 1: A *supervised classification learner* is a tuple $< I, O, F, U >$, 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.

Definition 2: An *empirical verification procedure* (EVP) is a tuple $< E, O, C_b, C_a >$ where O is a set of output symbols (output space) and E is an arbitrary, 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.

Any output space O of an empirical verification procedure is an empirically determinable set of equivalence classes. So, 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. In this paper we do not consider EVPs with non-unit cost.

Definition 3: An *Abstraction Network* (AN) is a tuple $< N, O, L, P >$, where N is a (possibly empty) set of ANs, O is a set of output symbols, L is a supervised classification learner, and P is an empirical verification procedure. 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.

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), a value determination must always be made by invoking P, which must be possible before classification in the case of AN leaves.

B. Object-level Reasoning

In a given problem instance, the values of the leaf nodes are fixed by observation. 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.

```
begin AN-reasoning($a$)
  1) If $a.N = \emptyset$, execute $a.P$ and return the result.
  2) Else, recursively execute this procedure for each $n \in N$ to generate an input string $i$ for $a.L$, then return $a.L.F(i)$ and store this value and $i$ for the purpose of the self-diagnosis procedure (called $a.last\_value$ and $a.last\_input$ below).
end
```

C. Meta-level Diagnosis and Repair

At some time after classification, the true value of the class label is obtained by the monitoring process. If the value produced by object-level reasoning was correct, no further action is taken. If the value is found to be incorrect, a self-diagnosis and repair procedure is followed. The specifics of this procedure are dependent upon the characteristics of the learner types that are used within nodes and the classification problem setting. For all of the empirical results detailed in this paper, the following procedure is used:

```
begin AN-diagnose-and-repair($a$)
  1) If $a.P == a.last\_value$ then return $true$.
  2) $\forall n \in a.N$, call AN-diagnose-and-repair($n$). If $\exists n \in a.N$ s.t. AN-diagnose-and-repair($n$) $== false$ then return $false$.
  3) $a.L \leftarrow a.L.U((a.last\_input, a.P))$, return $false$.
end
```

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.

Notice that an AN abstracts in two ways. One is apparent in the object-level reasoning procedure; information is progressively lost at each node in the hierarchy during reasoning as information is aggregated into equivalence classes, so abstraction takes place during inference. The second source of abstraction becomes clear in the self-diagnosis and repair procedure. The EVPs explicitly encode a process of abstraction from raw state to the equivalence classes produced at nodes in the AN.

The procedure detailed above is optimized to localize blame for classification errors using as few probes as possible under certain assumptions about error (no compensating faults) and the problem setting/learner type. A more conservative blame assignment strategy can be used when these assumptions are not reasonable – for instance, simply executing the EVP at each node in the hierarchy for every failure.

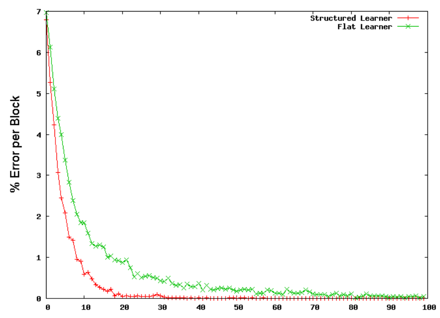
IV. Experiments

In order to verify that the diagnostic technique described above allows for correction of faulty knowledge in an AN, we have performed a set of experiments in a synthetic domain. The environment in this domain consists of a fixed abstraction hierarchy, 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 (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 and (4) performing structural credit assignment (SCA) and learning over the learner AN according to the procedures described above. EVPs within the inputs of both ANs are set up to quantize the floating point observations. EVPs are not needed at non-leaf nodes in the fixed AN, since no learning will occur. EVPs at non-leaf nodes in the learning AN are set up to examine the saved output value from the corresponding node in the fixed AN.

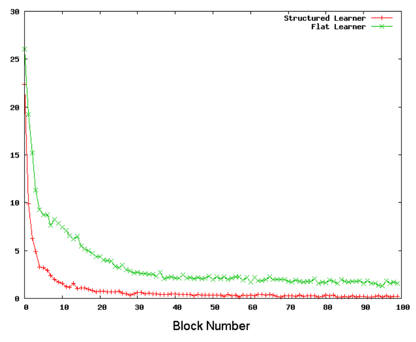
In addition to verifying that the diagnostic procedure described in this paper allows for correction of faulty AN content, we wished to determine the benefit of using a structured knowledge representation matching domain structure vs. using a *flat*, unstructured representation. Thus, in addition to 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 learner that must learn the full mapping from inputs to output values without the generalization enabled by a structured representation. In all experiments described here, we use simple rote learners within each node. When these rote learners receive a learning example, they merely directly record the example's mapping from input values to output, overwriting previous mappings if necessary.

In the experiments in the synthetic domain, all of the structured ANs take the form of binary trees (each non-leaf node has a fanin of two). Every node, including the leaves and the root, chooses from among 3 possible output values in this set of experiments. Thus, each rote learner used in structured learners in the synthetic domain has to learn $3^2 = 9$ mappings, while the flat learner must learn 3^{inputs}. We experimented with three problem sizes. The largest had 16 inputs, with the binary structure yielding 8, 4 and 2 nodes at each subsequent layer. The other two domains used 8 and 4 inputs, respectively. Since we train and evaluate the learners in an on-line, incremental fashion, we cannot apply the standard training set/test set approach to evaluation. Rather, we evaluate 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.

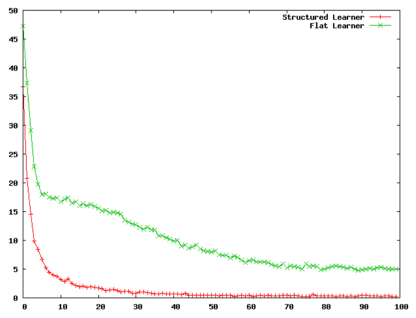
The results of these experiments for the three synthetic domain sizes are depicted in Figure 1 in terms of per-block error rate. The results shown are an average of 100 independent runs in each setting, with separate random rote learner initialization at the beginning of each run. Each run in the large problem setting consists of 10,000 generated examples, which we segment into 100 blocks of 100 games



(a) Layer sizes 4, 2, 1



(b) Layer sizes 8, 4, 2, 1



(c) Layer sizes 16, 8, 4, 2, 1

Figure 1. Per-block error rates of structured vs. flat learners for various problem sizes

each for the purposes of visualization. In the medium-sized problem setting, 100 blocks of 50 games were used in each run. Finally, in the small problem setting, each run consisted of 100 blocks of 10 games each. These results demonstrate the efficacy of the proposed method for SCA 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, so does the benefit of knowledge structure, as can be seen in these results.

A. Real Domains

We have also applied the above technique in real domains. In the domain of FreeCiv, an interactive turn-based strategy game, a game-playing agent must command units to build cities. To this end, the agent must make a series of crucial decisions as to whether the location on the game map currently occupied by a unit is suitable for the placement of a new city. We 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 2 illustrates a knowledge hierarchy for this task used by our FreeCiv agent.

For this experiment, we set up the AN depicted in Figure 2 with randomly initialized rote learners within each node. This AN was used to produce outputs from a set containing three values, corresponding to predictions of poor, moderate and good resource production for 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

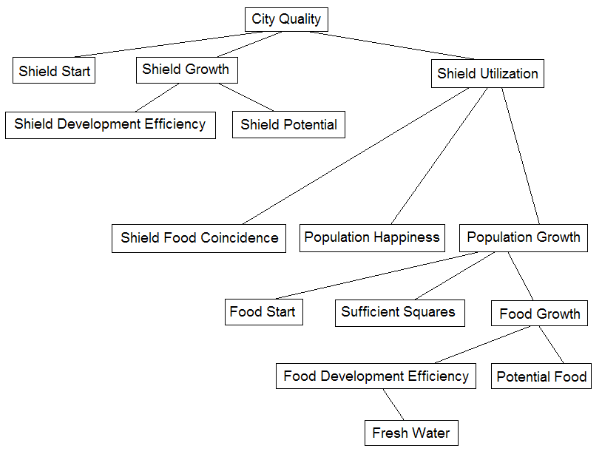


Figure 2. City Estimate Knowledge Hierarchy

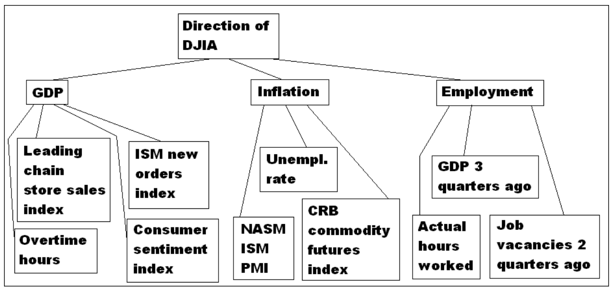


Figure 3. DJIA Abstraction Hierarchy

production. The empirical verification procedures simply discretize observed game features. All mappings of each rote learner were initialized to output zeros, which was known to be incorrect in some cases for each of the learners. In each trial, a sequence of games is run, and learning and evaluation occurs on-line. The learners are trained on sequences of 49 games. We segment these sequences of games into multi-game blocks for the purpose of evaluation; the block sized used is 7 games. Each game played used a separate 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. We observed a 52% decrease in the error rate of the learner, averaged over 60 independent trial sequences, when comparing the first block of examples to the 7th block.

In another, economic problem domain, we used the AN shown in Figure 3 to produce one of two values, DJIA-up or DJIA-down, corresponding to either a predicted rise (or lack of change), or a predicted fall in the value of the Dow Jones Industrial Average in the following month. We trained on monthly data from Jan 1960 - Nov 2005, yielding a total of 497 training examples. We again used the standard configuration of rote learners within nodes. We observed a 23.4% decrease in error between the first 213 and last 213 examples.

These preliminary non-synthetic experiments help to show that there is some more general applicability of AN-based learning beyond the artificial domain to problems of practical interest.

B. Integration with ANNs

ANs do not commit to rote learners within nodes, but rather can make use of a variety of supervised classification learning techniques within nodes. In order to demonstrate the generality of ANs with respect to the classification learners used within nodes, we will describe results obtained after integrating the AN framework with artificial neural network (ANN) code provided by Tom Mitchell and his students. This integration allows us to replace the rote learners used within AN nodes in most of the experiments described here with ANNs.

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 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[1]. Because the work described here is concerned only with repairing content and not structure, we do build the AN-ANN learner with correct structure that matches that of the fixed AN. 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 every example in the training set, perform inference with the fixed AN, saving the output values of all intermediate nodes and the root. Train both the AN-ANN and unaugmented ANN systems based on this inference over the fixed AN.
2) For every example in the test set, perform inference with the fixed AN, noting the value produced at the root. Perform inference with both the AN-ANN and unaugmented ANN systems, and determine whether the 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.

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 use randomly initialized rote learners within each node in the fixed AN, to simply provide a randomized mapping from inputs to outputs. Results obtained in a representative experiment are depicted in Figure 4. In this experiment, we use ANs with

[1]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.

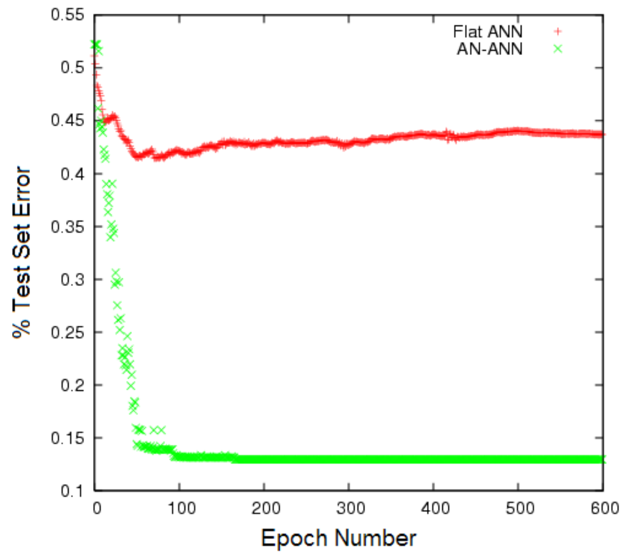


Figure 4. AN-ANN vs. Unaugmented ANN performance

a binary tree structure, with layer sizes 16-8-4-2-1. Each node is able to produce 3 values. The training set contains 1,000 examples, while the test set contains 10,000 examples. We ran the complete experiment 5 times (re-randomizing all learners and the fixed AN each time, etc), and Figure 4 depicts 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. A few other experiments with small parameter changes (e.g. hierarchy size) have been run, with similar 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.

V. Related Research

As we mentioned in the introduction, the use of metareasoning for self-adaptation in intelligent agents is related to learning: it views learning as deliberative, knowledge-based self-diagnosis and self-repair. In particular, our work on use of metareasoning for structural credit assignment in compositional classification is related to past work on tree-structured bias (TSB) [13][14]. In TSB, a concept hierarchy like those represented by ANs is used to limit the hypothesis space that must be searched by a learner. However, there are several *fundamental* differences between our work and past work on tree-structured bias. First, TSB has dealt only with binary classifications at all nodes in the hierarchy, while ANs can deal with multivalue classifications. Next, TSB research does not have the concept of EVPs, which encode the meta-knowledge used in our self-diagnostic procedure, instead relying on carefully constructed queries to the environment to learn the functions at internal nodes. Thus, rather than using explicitly represented meta-knowledge to perform self-diagnosis, TSB has a fixed training procedure that implicitly relies upon a given type of query. This procedure can be seen 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 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 [14], 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. The explicit representation of EVPs by ANs is also crucial to a major difference between AN research and past work on TSB. EVPs represent an abstraction from observable quantities to concepts used in an AN hierarchy. Because the grounding of concepts in observable quantities is explicitly represented, it becomes fair game to be operated upon during adaptation. It also means that we are able to adapt intermediate concepts themselves according to their functional roles – recognizing that intermediate concepts are not set in stone by the environment, but that they are constructs that exist in order to allow for correct overall classification.

VI. Conclusions

In this paper, we described a scheme for using metareasoning in intelligent agents for self-adaptation of domain knowledge. In particular, we considered retrospective adaptation of the content of intermediate abstractions in an abstraction network used for compositional classification when the classifier makes an incorrect classification. We showed that if the intermediate abstractions in the abstraction network are organized such that each abstraction corresponds to a prediction about percepts in the world, and meta-knowledge comes in the form of verification procedures associated with the abstractions, then metareasoning can invoke the appropriate verification procedures to perform structural credit assignment and adapt the abstractions. This provides credence to our hypothesis about the use of metareasoning for self-adaptation of domain knowledge: if the semantics of domain concepts can be grounded in predictions about percepts in the world, then meta-knowledge in the form of verification procedures associated with the domain concepts enables a metareasoner to address the structural credit assignment problem over hierarchies based on those concepts. We note however that whether this hypothesis holds for tasks other than compositional classification is an open question.

References

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

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

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

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

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

[6] J. W. Murdock and A. K. Goel, "Learning about constraints by reflection," *Canadian Conference on AI*, pp. 131–140, 2001.

[7] ——, "Localizing planning with functional process models," *ICAPS*, pp. 73–81, 2003.

[8] ——, "Meta-case-based reasoning: self-improvement through self-understanding," *J. Exp. Theor. Artif. Intell.*, vol. 20, no. 1, pp. 1–36, 2008.

[9] E. Stroulia and A. K. Goel, "Evaluating psms in evolutionary design: the autognostic experiments," *Int. J. Hum.-Comput. Stud.*, vol. 51, no. 4, pp. 825–847, 1999.

[10] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal*, vol. 3, no. 3, pp. 210–229, 1959.

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

[12] T. Bylander, T. Johnson, and A. Goel, "Structured matching: A task-specific technique for making decisions," in *Proceedings of the IEEE International Workshop on Tools for Artificial Intelligence, Fairfax (VA), USA*, 1989, pp. 138–145.

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

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