# Integrating Human Knowledge into a Relational Learning System

Dougal Sutherland*
Department of Computer Science
Swarthmore College
dougal@sccs.swarthmore.edu

**Abstract**

The real world can be successfully modeled as consisting of sets of objects, whose associated properties change over time. Relationships among these objects are critical to understanding the world, whether in classifying a current situation or in predicting its future behavior. An intelligent system, then, should be able to incorporate these types of relationships into its reasoning about the world.

This thesis presents an initial implementation of the Spatiotemporal Multidimensional Relational Framework (SMRF), first discussed by Bodenhamer et al. (2009). This probability estimation tree model has two primary contributions: it explicitly uses multidimensional relationships, whereas most relational learning approaches use only single-dimensional relations, and all of its relationships are dynamically learned from the data, rather than being defined *a priori* as in traditional relational learning.

The great flexibility of the SMRF model, however, necessitates expensive computations. To make learning more feasible for large and complex problems, we turn to advice from the human overseeing the learning task, who probably has some idea of what is important in the problem. We describe an early model of using this advice, supporting both advice that specifies objects as being important exemplars in the learning process and cues giving guesses about what kinds of properties are likely to be important to representing the problem. The two types of cues may also be combined.

We validate the effectiveness of both the SMRF model and its cue-based modifications in both two- and three-dimensional domains created in several different ways.

## 1 Introduction

The world is naturally represented as a set of objects, each of which has associated attributes. In order to reason effectively about these objects, though, one must often deal with relationships among them. These relations are key to many kinds of learning agents, from a domestic robot cooking a meal to a severe weather prediction system deciding whether a tornado warning should be issued for a particular storm.

One can also think of arbitrary predicates on a single object as a relationship of arity one; in this way, the *relational learning* problem is quite general.

This relational learning formulation has met with great success in many domains (Getoor & Taskar, 2007; Kemp & Jern, 2009). Much of that work, however, has operated in either the realm of graphical models or that of logical formulas. We choose an alternate path, that of augmented decision trees, which have the advantage of being relatively human-interpretable — a major boon in many use cases. The model presented here draws inspiration from the Relational Probability Tree model (Neville, Jensen, Friedland, & Hay, 2003) and its Spatiotemporal Relational Probability Tree extension (McGovern, Hiers, Collier, Gagne,

*Much of the work of this thesis was done with Matthew Bodenhamer, Thomas Palmer, and Dr. Andrew Fagg at the University of Oklahoma. See page 29 for details.

II, & Brown, 2008). These models, along with much other research in the field, are based on predefined categorical relations.

A key aspect of our approach is that in real-world situations, not all the objects at hand will actually be relevant to the concept being learned: a domestic robot making a sandwich need not worry about the oven. To keep from computing about more and more irrelevant aspects of the world, an intelligent agent must narrow its focus to only those objects that actually play a role in the situation. This problem, intimately related to the well-known *frame problem* of cognitive science, is as important for offline learners in complex domains as it is for the canonical robotic example.

It is obviously desirable in many systems to allow relations to be learned dynamically, rather than entirely predefined. The aforementioned models of Neville et al. (2003); McGovern et al. (2008) can define dynamic relations only by finding a split point along a single continuous dimension. Many interesting relations, though, are both continuous and multidimensional: color in an RGB space or Cartesian positions, for example. Moreover, these multiple dimensions may covary in interesting ways. For the "yellow" predicate in RGB space, for example, the blue attribute must be low, but the red and green values may range quite widely as long as their values are similar to one another.

The Spatiotemporal Multidimensional Relational Framework (SMRF), first defined by Bodenhamer et al. (2009) and also presented in Section 2 of this thesis, allows the user to define general *attribute spaces* within which it will search for interesting decision surfaces. A user would not define "yellow" as a predicate to be investigated, but rather "absolute color."

This generality, however, makes the search problem faced by SMRF rather daunting. It must decide which attribute spaces to consider, which objects to map in those attribute spaces, and also find a decision surface, potentially with quite large amounts of data. A successful and tractable search must therefore be constrained and guided in various ways.

Decision surfaces for a given attribute space may be found through an algorithm similar to Expectation-Maximization (EM), presented in Section 3. Because its results are extremely sensitive to initial conditions, we use the *Diverse Density* measure of Maron (1998) to determine good starting points for the search.

To further narrow down the search process, though, we must turn to other sources of knowledge entirely.

In many practical applications of the SMRF algorithm, there will be a human trainer facilitating the learning process. For a severe weather prediction system, this is the user who provides the original training process and supervises the learning. For a domestic robot, this might be the end user, or some other kind of teacher who guides the initial learning process.

These human trainers probably do not know the perfect representation of the concept at hand as a SMRF tree: if they did, they could just program it in manually. On the other hand, they likely do know something about the problem at hand. The meteorologist running the tornado prediction system might tell the system that wind speed is probably an important attribute to investigate; the domestic robot might be informed that the bread drawer is relevant to the sandwich-making task.

We model this knowledge transfer in two ways: *object cues*, which refer to specific objects in the world ("this object is important"), and *conceptual cues*, which instead refer to properties of attribute spaces ("color is important" or "blue is important"). The two can also be *mixed* ("it's important that this object is blue").

We can then use object cues to help with the search process for decision surfaces, whereas we use conceptual cues as a heuristic in picking the best attribute spaces to investigate. Details are presented in

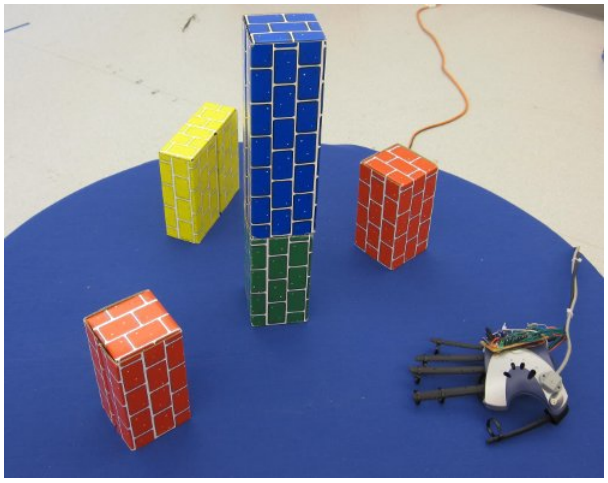


Figure 1: One simple example of the "blue above green" concept in the block world environment, with three distractors. The glove shown is used for data collection from real world examples, discussed in Section 5.3.

Section 4, including experimental evaluations.

Section 5 concludes and outlines directions for future work.

## 2 SMRF Trees

A SMRF tree is an augmented version of a *probability estimation tree* (Provost & Domingos, 2003). Given some collection of objects and their attributes, it provides an estimate of the probability that the collection contains at least one instance of its target concept. For example, in the block world domain in which most of this thesis operates, a particular SMRF tree might be used to recognize collections of objects which include a blue block on top of a green block (Figure 1).

In our model of the world, only some of the objects present in the environment are important to the concept at hand. Other objects are known as *distractors*.

For the purposes of this thesis, we work in a traditional block world domain. These domains have many advantageous characteristics for our work: there is little ambiguity in the definition of objects, and their attributes are relatively simple (we work mostly with position and mean color, although orientation, shape, and size information is also readily available). There is also a rich space of possible concepts: stacks of blocks with particular colors, triangular shapes, "goalposts," and much more.

There are many available data sources for this environment: entirely synthetic (Appendix A), from simulation controlled by either users or programmatic agents (Section 3.2.2), or through users manipulating real-world block configurations (Section 5.3).

Formally, we represent a collection of objects as a graph $G_i$, with the full dataset comprised of all the graphs denoted by $\Gamma$. Each graph consists of objects and their attributes.[1] A graph is denoted *positive* if it contains an instance of the target concept, and *negative* if it does not. We refer to the set of positive graphs

[1] The edges of the graph are the various relations which may be defined among the objects, based on their attributes. It would perhaps be preferable to use different terminology, but we use "graph" for consistency with the literature.

as $G^+$ and the set of negatives as $G^-$.

Given a dataset $\Gamma$, we wish to build a model that can (a) predict whether a novel graph would be positive or negative, (b) identify those objects from a given graph that are important to the target concept, and (c) provide a human-interpretable view of what the key features of the concept are. These tasks are inherently interconnected, and are treated as such in the SMRF model.

This task can be represented as a *multiple instance learning* (MIL) problem (Dietterich, 1997). In general, multiple instance learning is the situation where training labels are not given for items but rather for *bags* of items: a set of items is known to contain at least one positive example, but which example makes it positive is unclear. Bags may also be marked as negative, in which case it is known all members of the bag are negative.

The original motivating example for MIL is that of drug activity prediction: molecules may take on many conformations, only some of which can bind into a desired binding site. It would be extremely difficult, physically, to determine which conformation has the desired behavior; the MIL problem is thus to treat the entire set of conformations as a positive bag, with the conformations of other drugs that share the desired property as other positive examples, and those that lack the property as negative bags. The learning algorithm can then determine which conformations are the desired ones (Dietterich, 1997).

In our learning task, we label the set of all possible sequences of objects drawn from that sequence. That is, we label the graph of Figure 1 as positive. The MIL bag then consists of "items" which are sequences of any number of objects from the original graph, including $B, BG, GB, R_lY, GYR_rB$, and so on, where $B$ represents the blue object and $R_rY$ represents the sequence (red object on right, yellow object). These sequences are called *instantiation sequences* in the SMRF terminology.

It is worth noting that any given SMRF tree for the "blue above green" concept will choose either the $BG$ or $GB$ instantiation sequence as positive, and the other as negative; the choice is arbitrary. For certain kinds of concepts, the two sequences would both be positive examples.

## 2.1 Model Description

For clarity's sake, we describe the SMRF model first in terms of using an existing tree to classify a novel graph as exhibiting or not exhibiting the tree's concept.

A SMRF tree operates similarly to a classical decision tree. As evaluation proceeds, a set of instantiation sequences propagates down the tree, which contains four types of nodes:

**Root node** The root of the tree, where evaluation of a graph begins.

**Instantiation node** Extends each of this node's set of instantiation sequences by adding a new item from the graph, or a dummy item if the sequence already contains all items from the graph.

**Question node** Sorts the instantiation sequences that have reached it by a *mapping function*, to be described in more detail shortly. This node has three children: the *Yes branch*, for which the mapping function returns a positive result; the *No branch*, for which the mapping function returns a negative result; and the *Error branch*, for which the mapping function produces an error (as would a color question on an item that lacks a color attribute, or any question on a dummy item).

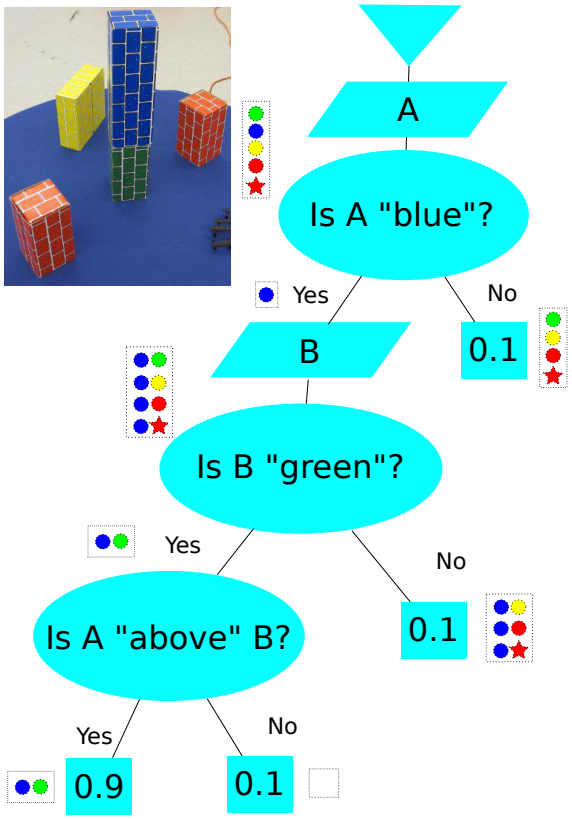



Figure 2: The evaluation of a novel graph (from Figure 1) by a SMRF tree for the concept "blue above green." The collections of instantiation sequences are shown near each node as evaluation progresses, with sequences running left-to-right; the red circle and red star represent the two red objects on the table.

**Leaf node** These nodes stop evaluation and assign a probability of exhibiting the target concept to each sequence that has reached it.

Consider the tree of Figure 2. The inverted triangle represents the root node, whose collection of instantiation sequences consists of the empty sequence from the graph.

Evaluation then progresses to the instantiation node marked A. This node adds each of the objects from the graph to the collection of instantiation sequences and passes the new collection (of five sequences) to the question node below. We refer to the object in this position as A.

The first question node then asks whether the object in the position labeled A is "blue." This actually means that the tree asks whether the color property of the object A falls within a volume in RGB-space stored within this node (to be discussed in more detail shortly). Sequences with this property go down the Yes branch, while others go down the No branch. Any sequences whose A object did not have a color property would go down the Error branch, not depicted in Figure 2 for space reasons.

The B instantiation node then extends the single $B$ instantiation sequence with each of the still-available objects in the graph. Had there been another sequence in the incoming collection, it would also be extended.

When a sequence reaches a leaf node, a probability is assigned that represents the tree's belief about the probability of that sequence exhibiting the target concept. In this case, the 0.1 probability for negative sequences and the 0.9 probability for the positive sequence represents uncertainty about having the right definitions of the concepts "blue," "green," and "above"; a learned tree of this form, though, would have probabilities much closer to 0 and 1.

Once we have propagated down the entire tree, each sequence will be assigned a probability by the leaf node to which it is sorted. The maximum such probability is returned as the overall probability that the graph exhibits the target concept.

The advantage of considering instantiation sequences is that we avoid the combinatorial explosion of considering all sequences of a given length from the graph; we can quickly prune sequences which are not useful down the No branches. In Figure 2, for example, we end up considering eight instantiation sequences at the leaves. Were we to consider all sequences of length two, we would have twenty sequences. The savings become even greater with more complex concepts and in the presence of more distractor objects.

### 2.1.1 Question Models

The question nodes of SMRF trees, as mentioned previously, are not predefined and as such do not have names like "blue" or "above" as used in the preceding description. They are instead defined by an attribute space and a volume within that space. The attribute space for "blue" would be an absolute color, and for "above" would be the relative location of two objects.

An attribute space is defined by a mapping function, which consists of essentially two parts: a *mapping parameter* and a *relational function*.[2] For the question model described above as "A above B", the mapping parameter is the sequence (A B), and the relational function defines the *relative location* concept of the "above" portion.

The mapping parameter defines which objects from the instantiation sequence are to be considered. This allows us to succinctly define the difference between, e.g., "A above B" and "B above A." Those two would

[2]The formal definition of model functions is somewhat more complex. We use a simplified description here, as it is more than sufficient for our purposes.

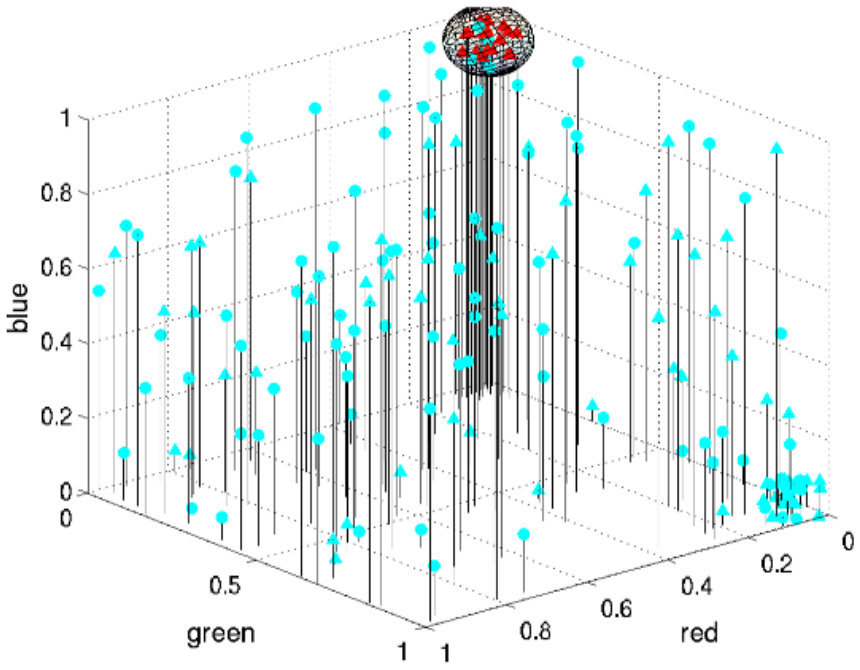



Figure 3: The "blue" question, shown as an ellipsoid in its "absolute color" attribute space. Each point represents an evaluation of an instantiation sequence from a training set of many examples, and is plotted according to the color of the sequence's $A$ object. Instantiation sequences from positive graphs are triangles, and negatives are circles. (The triangles scattered throughout colorspace are thus distractors from positive graphs.) The ellipsoidal region in the low-red, low-green, high-blue region is the decision surface; the points within that ellipsoid are considered "blue" and would go down the Yes branch, while other points would go down the No branch. (The cluster of points in the bottom-right are green objects.)

have different mapping parameters but identical relational functions.

The relational function, which represents a concept such as "relative location," consists of an *attribute template* and a *model function*. The attribute template defines which attributes to consider from each object, e.g. (location location) for "above" or (color) for "blue." The model function then determines how to combine these attributes: it would be the vector subtraction function for "above" and the identity function for "blue." We will generally refer to relational functions by simpler names such as "relative location" or "absolute color."

Now that we have a definition of the attribute spaces, question nodes must still define their volumes within that space – moving from "relative location" to "above." We choose to represent such volumes by the combination of a probability distribution function (pdf) and a threshold on that function. Ellipsoidal volumes are convenient in color or location spaces, and so a multivariate Gaussian is used. For orientation spaces, various other kinds of pdfs would be appropriate: Von Mises for 2D orientation, Dimroth-Watson for 3D, and the Girdle distribution for 3D orientations with rotational symmetry.

An example of the attribute space and volume of the "blue" question is shown in Figure 3.

# 3 Learning SMRF Trees

Now that the SMRF model has been defined, we define a method for learning SMRF trees from data. Given a dataset $\Gamma$ which distinguishes its graphs according to a concept, we wish to build a SMRF tree which will assign high probability to graphs which exhibit the target concept and low probability to graphs which do not.

The method described here is, as mentioned previously, an initial attempt at a learning algorithm for the SMRF model. Problems with the algorithm have necessitated investigation into an alternate learning algorithm, which will be sketched in Section 5.2. Nonetheless, this approach works well for some problems, and contains many of the key ideas which will be used in new approaches.

## 3.1 The Algorithm

We seek the tree which maximizes the likelihood of correct classification $L$, defined by

$$L = \prod_{G_j \in G^+} \max_{I \in {}^{\mathcal{L}}\mathcal{I}^j} \Pr(\mathcal{L}(I)) \prod_{G_j \in G^-} \max_{I \in {}^{\mathcal{L}}\mathcal{I}^j} (1 - \Pr(\mathcal{L}(I))), \tag{1}$$

where $G^+$ are the positive and $G^-$ the negative graphs, ${}^{\mathcal{L}}\mathcal{I}^j$ denotes the set of all instantiation sequences from graph $G_j$ which reach any leaf, $\mathcal{L}(I)$ is the leaf to which instantiation sequence $I$ is sorted, and $\Pr(\cdot)$ represents the probability assigned by a leaf.

Equation 1 has the intuition that the likelihood of the data given the tree is higher when instantiation sequences from positive graphs are sorted into leaf nodes with a high probability, and lower when no instantiation sequences from a positive graph are sorted into a high-probability leaf. Conversely, the likelihood of the data is also higher when no instantiation sequences from negative graphs are sorted into high-probability leaves, and lower when at least one such instantiation sequence is sorted into a high-probability leaf.

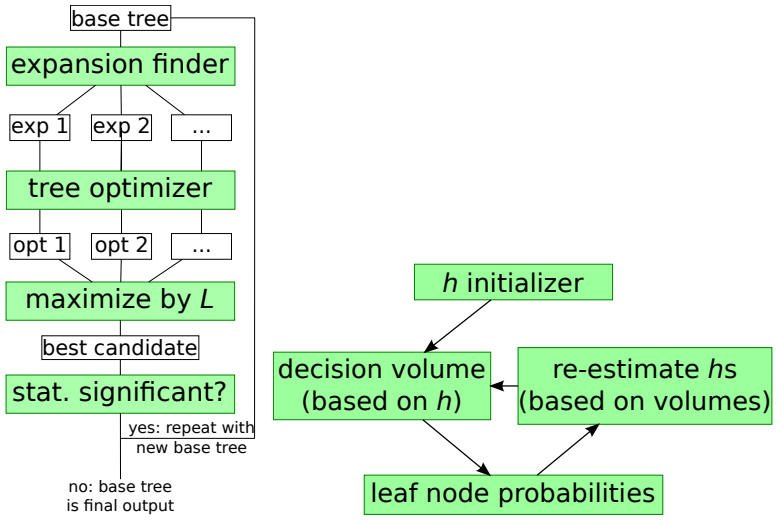
Following Zhang and Goldman (2001), we maintain a hidden variable $h$ for each instantiation sequence considered in the learning process. $h(I)$ represents the expected number of instantiation sequences that are descendants of the instantiation sequence $I$ that exhibit the target concept. For instantiation sequences at the leaves, then, $h(I)$ is the probability that $I$ itself is an example of the target concept, and must be between 0 and 1.[3]

### 3.1.1 Tree Expansion

Broadly speaking, tree expansion occurs as diagrammed in Figure 4a. We begin with a stub tree (simply a root node followed by a leaf node) and grow it.

When growing a tree, we consider the possible ways it might be expanded, and optimize each of those expansions. We then find the tree most likely to be better than the previous tree according to a statistical significance test. If it is sufficiently better, we accept that expansion and attempt to grow the tree again; otherwise, we take the previous base tree as the final output of the growth algorithm.

[3] In fact, $h(I)$ is the probability that $I$ represents the target concept *in the manner encoded by the tree.* In the blue above green example of Figure 1, (blue object, green object) might have an $h$ of 1 and (green object, blue object) an $h$ of 0.




(a) Tree growth. (b) Question optimization.

Figure 4: An overview of the learning process. We begin with a stub as the base tree in (a), find expansions, optimize (b), then find the best expansion and repeat. Note that in the actual algorithm, a single expansion may result in several different candidate optimized versions; see Section 3.1.2 for details.

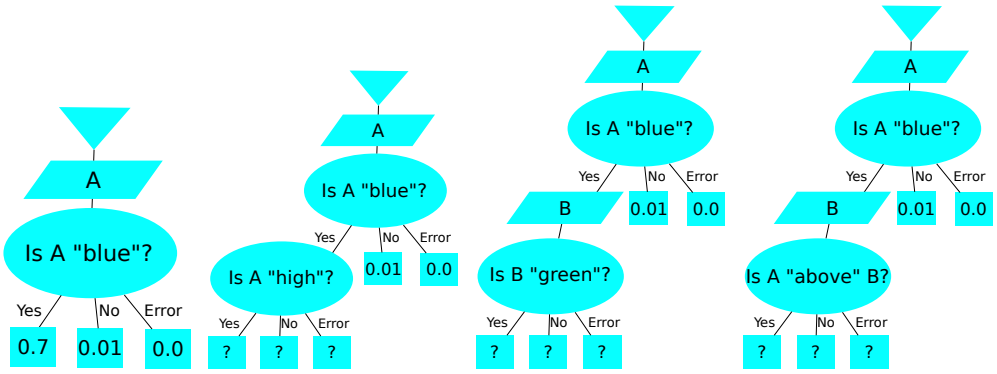
**Finding Leaves to Expand.** In expanding a SMRF tree, we replace a leaf node by some expansion, to be detailed momentarily. We wish to expand leaves which contain both "good" and "bad" instantiation sequences — it would be a waste of time to expand leaves that contain only one or the other. We thus determine which leaves are most likely to result in an improvement to the tree's likelihood metric (1).

To do so, we quickly find an optimistic splitting of the instantiation sequences at the leaf according to their $h$ values, and then calculate the $L$ of a tree which split instantiation sequences in that way. We determine the split by finding the point $\Theta$ at which the $h$s greater than $\Theta$ and those less than $\Theta$ are most likely to have come from different distributions, that is, the $\Theta$ for which the two sets have maximal Kolmogorov-Smirnov statistic. This provides a rough estimate of how separable the data is, according to our current beliefs about its true labels (Friedman, 1977).

Suppose we are expanding the tree shown in Figure 5a. We would identify the leaf down the Yes branch as a good candidate for expansion, since it contains a mix of positive and negative graphs (as can be seen by its 0.7 probability); the No and Error leafs would not be good candidates and would probably not be expanded.

**Generating Expansions.** Each possible expansion consists of replacing a leaf by some number of instantiation nodes, a question node, and leaf nodes as children of the question. The number of instantiation nodes inserted must be at most the arity of the question model; in fact, expansions which include instantiations not referred to by the question are illegal.

The possible mapping functions for a question node are defined by one of the user-defined relational functions (e.g. "relative location") and a mapping parameter determined by the shape of the tree. Mapping parameters referring to the most-recently and least-recently instantiated objects are allowed.




(a) The base tree. (b) A possible expansion. (c) A possible expansion. (d) A possible expansion.

Figure 5: An intermediate SMRF tree for the blue above green task (a) and some partially optimized expansions (b)–(d).

Some possible expansions to the tree shown in Figure 5a are shown in Figure 5b-d. Note that the leaf probabilities are unknown at this point. In fact, the expansions at this point would simply ask about "absolute location," "absolute color," and "relative location," respectively; the "high," "blue," and "above" interpretations only appear once the question optimization phase has been entered.

Once the set of possible expansions has been generated, the original SMRF algorithm considers all of them, or perhaps a random subsample. This behavior is one of the major changes made with the addition of cues, as described in Section 4.2.

**Evaluating Expansions.** Each candidate expansion is fully optimized according to the algorithm described in Section 3.1.2 below. If the expansion contains an instantiation node, the optimization is repeated from multiple starting points in the space of $h$ assignments; the sampling method is described below.

Once the tree $T_i$ with the highest likelihood $L(T_i)$ has been determined, we must decide whether the improvement in likelihood from the base tree represents a statistically significant improvement or is now simply overfitting the data. We do so through a likelihood ratio test, where the likelihood ratio is $\Lambda = L(T_i)/L(T_{i-1})$. For reasonably large sample sizes, $-2\log(\Lambda)$ is approximately $\chi^2$ distributed, with degrees of freedom equal to the number of parameters added by the expansion (Huelsenbeck & Crandall, 1997).

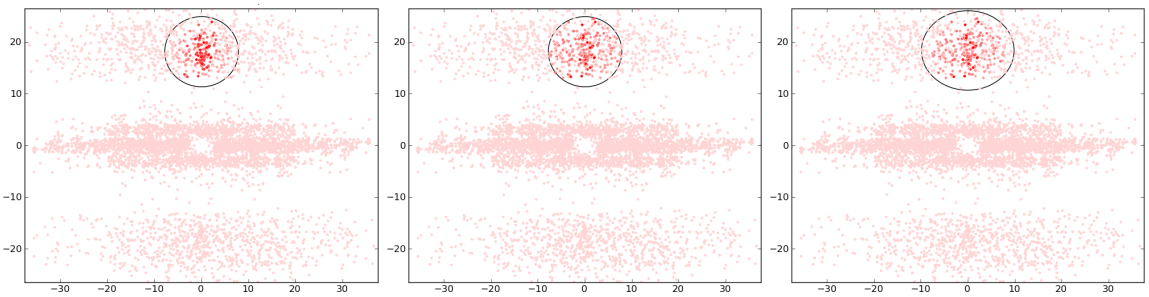
We calculate the number of parameters in a tree as follows:

- 1 for the probability stored at each leaf node
- 1 for each instantiation node
- length of the mapping parameter plus the number of distribution parameters for each question node

The tree of Figure 5b thus has 26 degrees of freedom: 5 leaf nodes, 1 instantiation node, 3 for the mean and 6 for the covariance matrix at each of the 2 question nodes,[4] and 1 each for the mapping parameters of "blue" and "high." This is 12 more than the 14 of Figure 5a, so the $\chi^2$ test would use 12 degrees of freedom.

Note that the calculation of the number of parameters in a tree is somewhat arbitrary; we have merely chosen a method that seems reasonable.

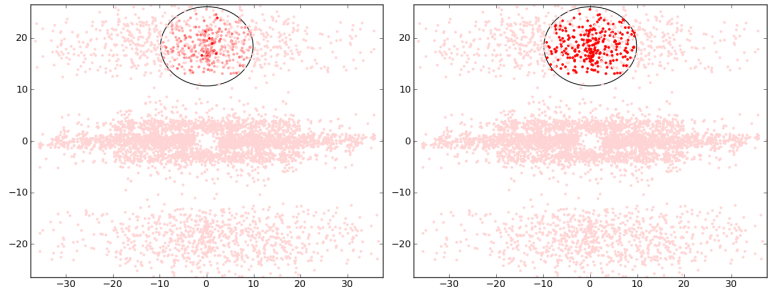
[4]Covariance matrices are symmetric, so there are 6 unique elements in a $3 \times 3$ matrix.



(a) After $h$ initialization and the first decision surface step.

(b) After the first hidden variable estimation.

(c) After the next decision surface step.



(d) After the next hidden variable estimation.

(e) After the final, global hidden variable estimation.

Figure 6: Optimizing a two-dimensional relative location question (from Section 3.2.2). The red dots correspond to instantiation sequences, with the intensity of the red corresponding to $h$ (darker corresponds to higher $h$). The axes are relative $x$ and relative $y$ position of the two elements in the sequence.

Since we consider several expansions to the tree, we must account for the *multiple comparisons problem* (if we make several evaluations, the probability of mistakenly calling an expansion significant when it is not[5] in any one of the evaluations increases). We do so with the Bonferroni correction, so that an expansion is considered significant only if $p \leq \alpha/N$, where $N$ is the candidate number of expansions. We use a significance level of $\alpha = 0.1$ in this work.

### 3.1.2 Question Optimization

Given a newly expanded tree, we need to find an optimal volume for the new question node's model. We do so using an approach similar to Expectation-Maximization (EM), iterating between redefining the volume based on $h$ values and redefining $h$ values based on the volume until convergence.[6]

As mentioned previously, we seek the distribution parameters and a threshold for a pdf, the type of which is predetermined according to the kind of question we are optimizing. For an absolute color question as in Figure 3, we are finding the mean and covariance matrix of a 3D Gaussian, along with a threshold on that Gaussian, which defines an ellipsoid.

[5] That is, mistakenly rejecting the null hypothesis; a Type I error.

[6] This process is not actually EM, as the metrics by which we optimize in each step are not identical, and hence convergence is not guaranteed. Using only the metric $L$ would be desirable; work is ongoing to remedy that problem, as described in Section 5.2.

**Initializing $h$ Values.** Our search process, like most EM searches in non-convex spaces, is quite sensitive to its starting point. A search that begins its optimization from a question volume which covers the entire space, for example, will almost never make much progress.

We therefore find a good starting configuration for the search by intelligently initializing the $h$ values of new instantiation sequences. We use the *Diverse Density* (DD) measure of Maron and Lozano-Pérez (1998); ? (?) to find a good starting configuration. DD is standard in multiple-instance learning problems. Recall the MIL view of our task from page 4: we are given sets of positive and negative "bags," where the items in each bag are its instantiation sequences at a given point in the tree. We thus wish to find an estimation of the probability that each item is the key element of the bag which makes it positive: that is, its $h$ value. This is exactly the problem that DD solves.

When optimizing a question, we consider the positions of each instantiation sequence in the attribute space at hand. Optimizing the "blue" question of the tree in Figure 2, we seek areas of the color space where each positive example contains at least one nearby instantiation sequence, and negative examples do not.

Formally, we use the most-likely-cause estimator of DD for a point $t$ in the attribute space of node $q$:

$$\widehat{DD}(t) = \prod_{G_j \in {}^qG^+} \max_{I_i^j \in {}^qG_j} \left[ \Pr\left( I_i^j \in c_t \right) \right] \prod_{G_j \in {}^qG^-} \left( 1 - \max_{I_i^j \in {}^qG_j} \left[ \Pr\left( I_i^j \in c_t \right) \right] \right), \tag{2}$$

where ${}^qG^+$ and ${}^qG^-$ are the positive and negative graphs with any instantiation sequences at question node $q$, ${}^qG_j$ are the instantiation sequences from graph $G_j$ which reach node $q$, and $\Pr\left( I_i^j \in c_t \right)$ represents the probability that instantiation sequence $I_i^j$ matches the concept defined by $t$ — in this case, the probability that the color of its first object is fundamentally similar to the point $t$ in color space.

Given a point $t$ in our attribute space, then, (2) says that we consider only the instantiation sequence from each bag whose position in attribute space is nearest to $t$. DD is maximized when more positive bags have nearby points, and when fewer negative bags contain points near $t$.

We follow Maron (1998) in calculating $\Pr\left( I_i^j \in c_t \right)$ through a Gaussian-like probability calculation (that is, a multivariate Gaussian normalized to lie in the range [0,1] rather than to have total probability mass of 1). The covariance matrix of this Gaussian is calculated by computing the variance of the positive instantiation sequences at this node along each dimension of the attribute space, and multiplying by a "kernel factor" (generally 0.2 in this work). The matrix is thus constrained to be diagonal. Empirically, the quality of results in the DD calculation phase is sensitive to the kernel factor; the appropriate factor seems to vary across different problems. Further investigation into methods for setting an appropriate kernel factor is needed.

Calculation of DD according to (2) is a relatively expensive process; to perform it at many points $t$ in the attribute space would be prohibitively time-consuming. Fortunately, though, points of maximum DD should lie near each positive bag, and so we can constrain our search by only evaluating DD at each of the instantiation sequences from a small number of positive bags (2 in the experiments here). Of those points, we find a small number of maxima (2, 4, or 8), though we avoid wasting computation from choosing points which are extremely close together (and so will result in near-identical optimizations).

Once we have picked a mean for the initial $h$ assignment, we calculate initial $h$ values for all new instantiation sequences according to their distance from the mean. Instantiation sequences are grouped by

ancestors with fixed $h$, and then $h$ mass is distributed among descendants proportionally to their pseudo-Gaussian evaluation of similarity to the mean as described above.

**Selecting the Decision Surface.** Assuming that we have $h$ values for the instantiation sequences reaching question node $q$, we wish to select model parameters for $q$ which will separate instantiation sequences with high $h$ from those with low $h$.

First, we find parameters $\theta_q$ for the model pdf through weighted maximum likelihood estimation. Specifically, we maximize the *model data likelihood* $\hat{L}$:

$$\hat{L}(q) = \prod_{I \in {}^q\mathcal{I}} p\left(\phi_q(I) \mid \theta_q\right)^{h(I)}, \tag{3}$$

where $\phi_q$ is the mapping function of node $q$, and $p$ its associated pdf. The intuition for (3) is that instantiation sequences with high $h$ values should contribute strongly to the selection of the parameters, while those with low $h$ should contribute little. A closed-form solution for the multivariate normal distribution exists and is analogous to the unweighted version; some other pdfs, such as the Von Mises distribution, must be fit iteratively.

Once we have pdf parameters $\theta_q$, we also need a threshold $\Theta_q$ to define our decision surface in attribute space. Instantiation sequences $I$ with $p(\phi_q(I) \mid \theta_q) \geq \Theta_q$ are sorted down the Yes branch, and others down the No branch. The threshold value $\Theta_q$ is set to the point with maximal Kolmogorov-Smirnov distance between the cumulative distributions of the $h$ values and the $1-h$ values — that is, the point at which the distributions of $h$ and $1-h$ are most likely to be underlyingly different, which Friedman (1977) establishes as a good metric.

**Finding Leaf Node Probabilities.** We next must calculate leaf node probabilities. After sorting instantiation sequences down the tree, we wish to maximize the likelihood of correct classification $L$ (1). Note that this task is made more difficult because $L$ involves taking the maximum of leaf probabilities.

If we consider a leaf with probability $x$ to which $p$ positives and $n$ negatives are sorted, its contribution to $L$ will be $l(x; p, n) = x^p\,(1-x)^n$.

For fixed $n$ and $p$ and for $0 < x < 1$, we can use simple calculus to find $l$'s sole critical value (which is a maximum) to be at $x = \frac{p}{n+p}$. Furthermore, when both $n$ and $p$ are positive, the endpoints 0 and 1 have value zero; if either $n$ or $p$ is zero, then the preceding formula obtains the endpoint as the maximum. If both are zero, we have $l(x; 0, 0) = 1$, which does not contribute to the overall likelihood whatsoever; we arbitrarily choose to use $x = 0$ in this case.

Pleasingly, this result is exactly the intuitive frequentist value for the probability at the given leaf.

We can now maximize $L$ by assigning leaf probabilities according to Algorithm 1, which has asymptotic runtime approximately $\mathcal{O}\left(|B|\,|L|\right)$.[7] Leaves are considered in descending order of their final probabilities, so that for each leaf, we determine with certainty which bags would be assigned to it by the max operators, and then maximize $L$ with respect to those bags.

[7] The outer loop runs $|L|$ times, and the inner loop has $|L|$ executions each of about $|B|\,/\,|L|$ time, assuming we have a mapping from leaves to instantiation sequences at each leaf.

**Algorithm 1** Leaf probability assignment.

$B \leftarrow$ the set of all bags (instantiation sequences grouped by graph)
$L \leftarrow$ the set of all leaf nodes
**while** $B$ not empty, $L$ not empty **do**
  **for** $l \in L$ **do**
    find each $b \in B$ with an IS at $l$
    compute $p/(n+p)$ with respect to these $b$s
  **end for**
  $max\_l \leftarrow$ the $l \in L$ with the maximum ratio
  $max\_l$.probability $\leftarrow$ $max\_l$'s ratio
  remove $max\_l$ from $L$
  remove the $b$s which reach $l$ from $B$
**end while**
set the probability of each $l \in L$ to 0

**Hidden Variable Estimation.** After optimizing the question parameters and calculating leaf node probabilities, we need to select a set of $h$ values maximally consistent with the tree's sorting of the data. We do so by maximizing the likelihood of correct sorting $\bar{L}$:

$$\bar{L} = \prod_{I \in \mathcal{L}\mathcal{I}} \Pr\left(\mathcal{L}(I)\right)^{h(I)} \cdot \left(1 - \Pr\left(\mathcal{L}(I)\right)\right)^{(1-h(I))}. \tag{4}$$

$\bar{L}$ thus favors trees which sort high-$h$ instantiation sequences into high-probability leaves, and low-$h$ instantiation sequences into low-probability leaves. In the optimization process during which leaf probabilities are held constant, $\bar{L}$ is optimized by assigning low $h$'s to instantiation sequences which sort to leaves with probabilities below 0.5, and high $h$'s to those which obtain probabilities above 0.5.

Solving for the $h$'s becomes a linear programming (LP) problem, in which $\log \bar{L}$ is maximized subject to the following constraints:

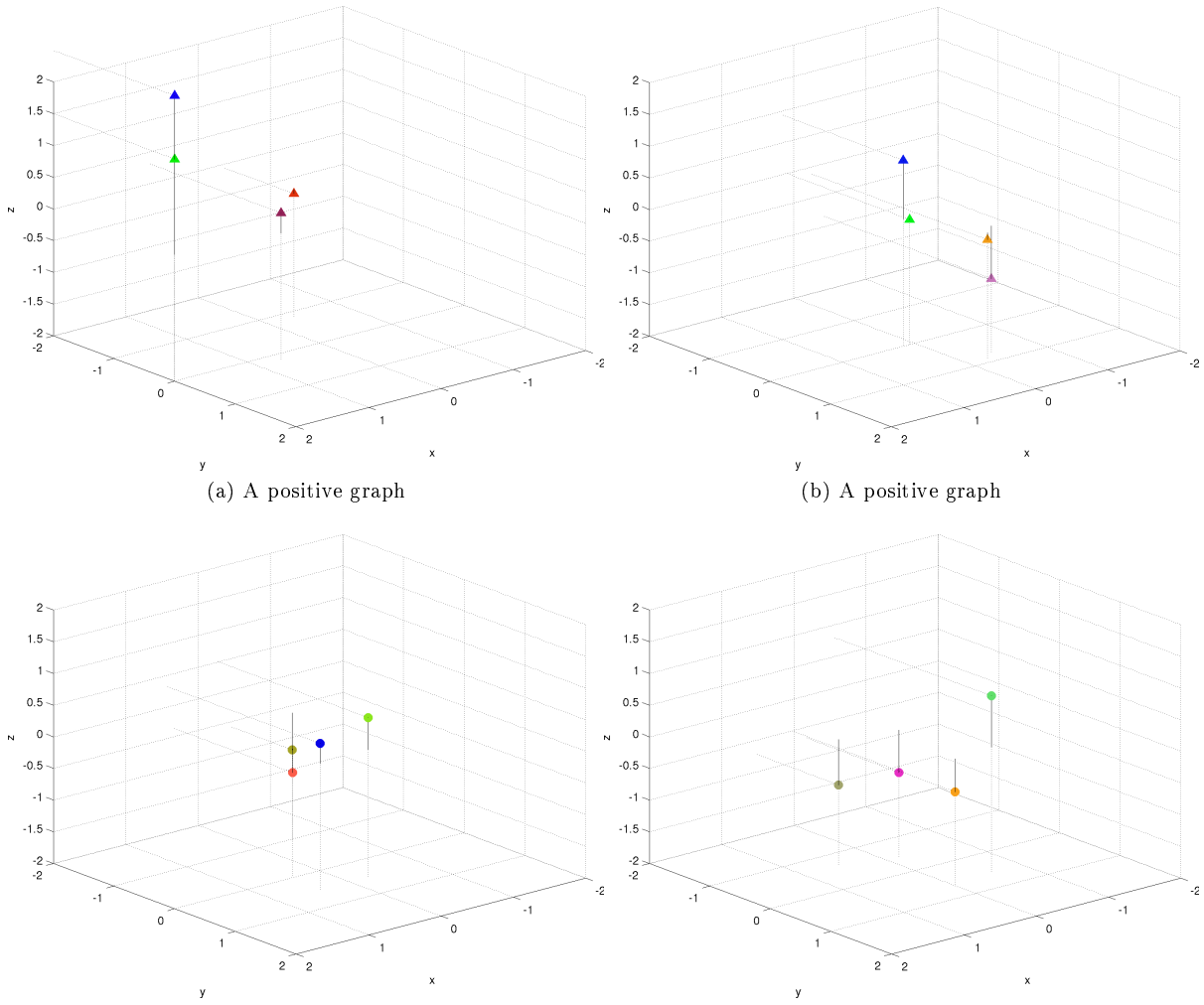
- $0 \leq h(I) \leq$ the number of instantiation sequences in $I$'s graph
- $0 \leq h(I) \leq 1$ for $I$s which reach a leaf
- $h(I) =$ the sum of the $h$'s of $I$'s children for non-leaf $I$s
- the root of a positive graph has $h \geq 1$
- $I$s from negative graphs have $h = 0$

**Repeat.** After optimizing the $h$ values, the new $L$ is calculated and compared to its value under the previous parameters. If the difference is small enough, we stop iteration; otherwise, we go back to re-estimate pdf parameters and proceed from there.

When evaluating an expansion, the process above is first applied for instantiation sequences local to the new question, then globally; the hierarchical process is then repeated. This allows the algorithm to select a proper question model locally before adjusting other models to account for the new one.

## 3.2 Experiments

We present here experimental results that demonstrate the algorithm is able to successfully create models for various target concepts, including fairly complex multi-question concepts, even in the presence of many



(a) A positive graph

(b) A positive graph

(c) A negative graph

(d) A negative graph

Figure 7: Typical examples for the blue above green dataset.

"distractor" objects and with noise in the labels of the training examples.

### 3.2.1 Synthetic Multi-Object Structures

We first test that the learning algorithm can successfully model block world structures of moderate complexity. We model the blue above green example discussed previously, and generate examples like those in Figure 7 using the system of Appendix A. The generated dataset consists of 100 positive and 100 negative examples; all experiments presented here (and elsewhere in this thesis) use tenfold cross-validation.

These data sets consist of objects with two kinds of attributes: 3D position and RGB color. For the position attribute, the algorithm could choose to model the object's absolute location in $\mathbb{R}^3$, the scalar distance between two object locations, or the vector difference between two object locations in $\mathbb{R}^3$. For color attributes, the algorithm could choose to model either the RGB color of an object or the difference vector in

RGB space between the colors of two objects. The pdfs used for all of these mapping functions Gaussians of the appropriate dimension (one for scalar distance, three otherwise).

The first experiment with the blue above green dataset examines the learning algorithm's performance in the presence of two kinds of interference: distractor objects and noise in the labels. Distractor objects are other objects present which do not participate in the target concept, such as the red and yellow objects in the situation of Figure 2. We implement noisy labels by "corrupting" the labels of a uniformly random subsample from the set of training graphs; we simply switch it from being counted as a positive example to a negative one, or vice versa.

Because SMRF trees assign a probability rather than a classification decision, we use the area under the receiving operating characteristic curve (AUC) to measure performance (c.f. Hastie, Tibshirani, & Friedman, 2009). An AUC of 0.5 represents completely random performance, while a 1.0 represents a classifier which assigns probabilities of 0 and 1 with perfect accuracy.

The means and standard deviations of the AUCs obtained for various numbers of distractors as a function of corruption are shown in Figure 8a. We also tested the effect of training set size on learning performance when corruption is not present; those results are shown in Figure 8b.[8]

Generally, the algorithm performs better with lower numbers of distractors, as the positive and negative examples are more easily separable. Interestingly, we see only small effects on performance for low levels of graph corruption, and then a sharp drop-off in performance to random behavior (where the final tree is simply a stub). The drop-off point varies depending on the number of distractors; for 10 distractors, it appears to be between 15 and 20% of labels corrupted on this dataset, while it is between 25 and 30% for 0-5 distractors. This shows that the learning algorithm is robust to even significant amounts of noise in the training data labels, and is more resilient to this noise when the data itself is clearer (i.e. there are fewer distractors present).

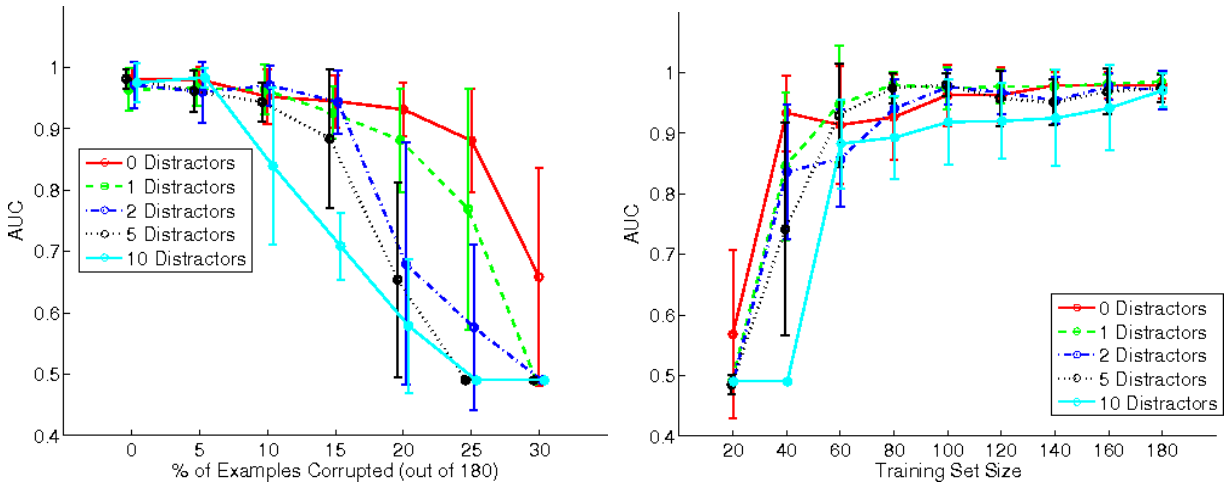
A typical tree learned by the process, particularly for low numbers of distractors and corruption, resembles that of Figure 2. Many trees, though, rather than modeling the absolute color of each of the green and blue blocks, choose to ask about the vector between the colors of the two blocks. Because of the shape of RGB space, these two strategies are essentially equivalent. (If the color means were closer together, these questions would not be equivalent, and the learning algorithm would probably use the two-expansion form.) As the number of distractors and amount of corruption increases, however, we see a performance drop due to both fewer questions being asked and the decision surfaces at each question growing in size.

We can also see from Figure 8b that the learning algorithm's performance is near-maximal with only 40 training examples, for modest numbers of distractors. Being able to learn concepts with small amounts of data is important for real-world problems, particularly those with a robotics application.

#### 3.2.2 Predicting Action Outcomes

We also approach the problem of learning concepts which can predict world dynamics, an important application of SMRF for robotics tasks. We obtain data from a two-dimensional simulated block world environment,

[8] These results show up to 10 distractors. We have run experiments with 20 and more distractors, but the experiments take significant amounts of time, primarily because the method of calculating Diverse Density described in Section 3.1.2 is slow. Work is underway to speed up that process through approximation, as it seems unlikely that exact results for DD will be much more useful than approximate results. The results for 20 distractors have been behaved similarly to those with 10 distractors in past experiments.




(a) As a function of corruption.

(b) As a function of training set size.

Figure 8: Mean and std AUCs for the blue above green experiments.

in which blocks can be manually moved around and are subject to gravity.[9]

In this task, a programmatic agent grasps a random block, moves it to a uniformly random position at a specified height over the world, and drops it. The learning task is to predict whether the block will settle to be supported on another block in some way, or come to sit flat on the ground. Figure 9 shows an example of the former case in the simulator. This task is in some ways a structural relation similar to the blue above green task, but it is both harder to learn and provides an example of how SMRF might be used in more dynamic robotic environments.

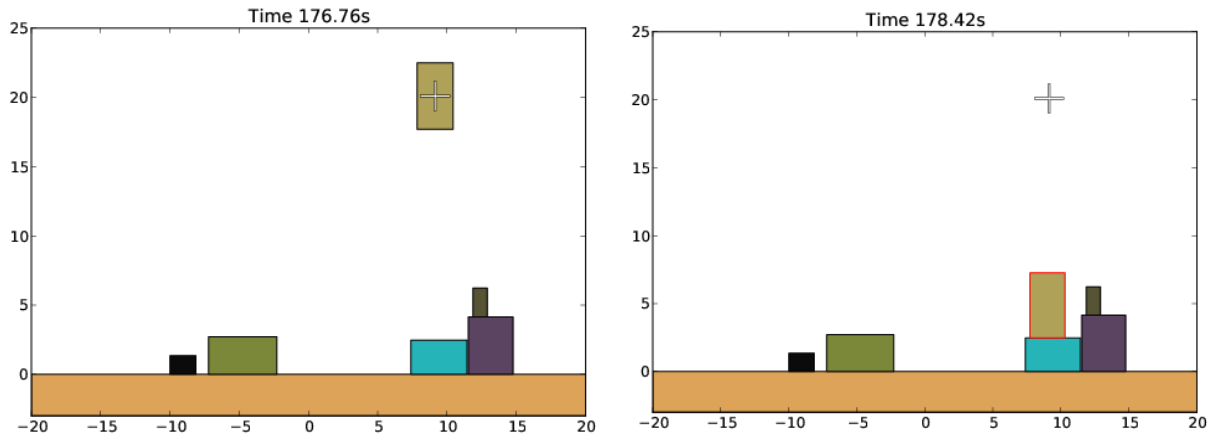
To gather data, the virtual dropping agent was run for one hour of simulation time, with all blocks being cleared every thirty seconds. After finding the world states and labeling them, we balance their numbers to have approximately as many positive as negative examples.

In these experiments, the only attributes available to the learning algorithm are centroid position and color. (We allow the same mapping functions as in the previous experiment, though those for locations operate in $\mathbb{R}^2$.) Clearly, however, size and shape information also play a large role in determining where the block will settle. Unlike the previous problem, then, perfect classification is simply not possible for this experiment.

For many parameter settings, the SMRF learning algorithm was not able to learn any statistically significant expansions to the stub tree. When it did, however, they resembled the tree shown in Figure 10, asking whether there are any pairs of blocks where one (the block being dropped) is in a given region above the other. Note that this question applies to *any* pair of blocks, and if there were a sufficiently tall tower of blocks off to the side, it might also satisfy the question. In general, a solution to this problem will require the addition of a categorical attribute marking the block to be dropped.

Figure 11a shows the performance of the learned tree for ten-fold cross-validation on the training set, with subsampling to various numbers of examples and with different portions of the example labels reversed.

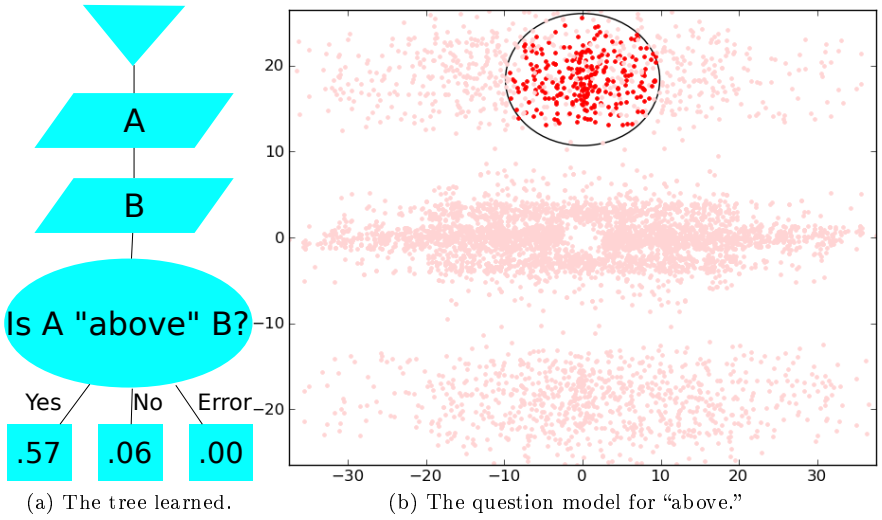
[9]The simulator, based on the JBox2D dynamics engine, is available at `http://code.google.com/p/stackiter/`.




(a) World state just before release.

(b) World state after the block has settled.

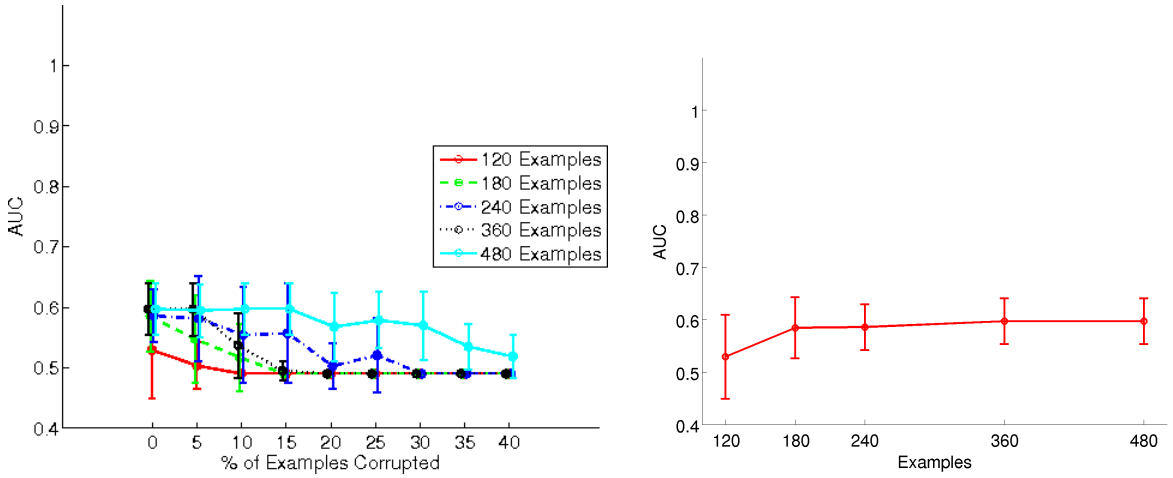
Figure 9: A view of the dynamics simulator for Section 3.2.2. The learning task here is to predict, given the state (a) just before release, whether the dropped block will end up resting on another block as in (b) or will settle on the ground.




(a) The tree learned.

(b) The question model for "above."

Figure 10: A typical tree learned for the dynamics simulation.




(a) As a function of the portion of labels corrupted.

(b) As a function of training set size, with no labels corrupted.

Figure 11: Mean and std AUCs for the dynamics experiments.

We see that, as expected, maximal performance on this dataset is much lower than that on the problems of the previous section. The maximal AUC observed is about 0.6, similar to what a hand-coded relative location question is able to reach on this dataset. This maximal value is essentially the same across different numbers of examples, and indeed they actually learn quite similar trees. The differences in AUC, then, primarily show the portion of runs which successfully learn a tree, as opposed to those that result in a just a stub.

With no labels corrupted, only the 120 example run performs worse than the others (shown more clearly in Figure 11b). As the portion of corrupted examples increases, however, Figure 11a shows that runs with larger training sets are more robust to noisy labels. This behavior is unsurprising, because the labels are inherently underspecified given the amount of data provided. Within the area in which the top block is more or less above another block, the corruption has little effect, as it is simply moving some portion of the examples from one label to the other. For low rates of corruption, then, only a few examples with no block underneath the dropped block will be switched; there will still be enough examples that it is profitable to ask that question. For smaller numbers of training examples, however, the point at which there are too many examples outside the desired region is lower.

# 4 Cuing

The representational flexibility of SMRF trees comes at a cost: an extremely large search space. For each node in the tree, the algorithm must consider the kind of question to ask and which items to ask it about (i.e. the mapping function), and then it must determine a volume in the defined multidimensional attribute space.

Indeed, the initial results in Section 3.2 demonstrate that the learning algorithm's classification performance and speed decrease both with added noise in the data and with more complex models. The search process becomes lost in poor areas of the model space, not having the right hints to bring it towards maxima.

But it is this very flexibility which is the strength of the model, and we do not wish to constrain the kinds of outputs that can be made. We therefore seek a way to guide the search. We already have a few such hints based on statistical properties, such as the Diverse Density technique of Section 3.1.2. But for further help in the search process, we turn to another kind of knowledge entirely.

In many domains, the learning process is managed by a human overseer. This trainer often has some knowledge of the problem domain (though not necessarily of the intricacies of the learning system). Can we exploit some of that domain knowledge to help guide the learning process?

## 4.1 Related Work

Most of the previous work on integrating human feedback into has taken place in a reinforcement learning setting, where humans can provide positive or negative reinforcement to the learner either in conjunction with a traditional Markov Decision Process (MDP) reward signal, or entirely replacing it.

For example, Knox and Stone (2010, 2011) integrate human feedback with an MDP signal. They describe the human feedback as informative yet potentially flawed, while the MDP signal is informationally poor (as rewards are usually delayed and rarely fully discriminate between good and bad actions) yet by definition flawless. They find that action biasing (adding a weighted prediction of human reinforcement to action selection) results in greatly improved performance as compared to standard reinforcement learning without human input.

Kuhlmann, Stone, Mooney, and Shavlik (2004) take a related approach to modeling human advice in a subproblem of the well-known RoboCup Soccer domain. Rather than providing direct scalar human reinforcement to their learners, however, they receive advice of the form "no one pass to the goalie," which are then translated through one of several learning-based mechanisms into a Lisp-like formal language of predicates and advice. During the agent's execution, predicates are matched against the game state and apply corresponding bonuses or penalties if they match.

The advantage of combining advice and learned values for actions in reinforcement learning is that bad advice can be effectively discarded, if it is learned to be bad. This is a highly desirable trait, as human advice is likely to be either inaccurate or inappropriate to an algorithmic view of the world in many situations.

Maclin, Shavlik, Torrey, Walker, and Wild (2005) extend the Knowledge-Based Kernel Regression method for giving advice to reinforcement learners. Teachers can provide situational preferences for actions similar in content to the formal language output used by Kuhlmann et al. (2004). These preferences are then optimized in the support vector machine learning function through linear programming. The presence of slack variables allows for some of the same ability to discard bad advice.

Thomaz (2006) reframes the problem as "Socially-Guided Machine Learning," focusing more on providing natural interactions with the human trainer. Thomaz provides three implementations of socially-guided learning tasks. The first is a guided planning-type algorithm. The second is a reinforcement learning approach similar to that of Kuhlmann et al. (2004) and Knox and Stone (2011). The final system combines the ideas from both, adding intrinsic motivation and several kinds of human interaction to a standard reinforcement learning system, including attention direction, action suggestion, goal state labeling, and positive and negative feedback. Critically, this version of the system supports natural interactions with human trainers, but can function normally with no human input whatsoever.

In a different domain, Walker et al. (2010) use human advice to set up the initial conditions of an

Inductive Logic Programming (ILP) system. Users provide information on why specific examples satisfy or do not satisfy the concept, which are then converted into background knowledge after a generalization process. Generally, however, research relevant to our current task in frameworks other than reinforcement learning appears to be somewhat limited.

## 4.2 Cuing Formulation

We choose to support two types of human advice in our initial implementation of human-provided cues within SMRF. They can theoretically serve the dual purposes of speeding up the search and allowing the algorithm to learn concepts which the uncued algorithm would not be able to learn. Broadly speaking, cuing is analogous to adding a good heuristic to search processes: the search space remains the same size, but the algorithm has hints about what parts of the space might be better, both speeding up the process and (for non-exhaustive searches) making certain solutions much more realistic to find.

*Conceptual cues* refer either to specific attributes or to a preexisting vocabulary of relations, e.g. "color is important" or "being blue is important." *Object cues* refer to specific objects in the world, e.g. "this block here matters." Cues may also be *mixed*: "it's important that this block here is blue."

Cues may be positive, as above, or negative: "don't worry about the red blocks." In fact, this initial formulation supports arbitrary scalar weights, though whether this flexibility makes sense in terms of user interaction is not yet clear.

These two types of cues cover a large portion of the intersection between the kinds of properties that a human would know about a dataset and the kinds of data which can be readily integrated into the SMRF framework.

In the domestic robot case, object cues might focus a robot on only those objects actually important to the task at hand: when teaching a robot to make a sandwich, these cues might focus the robot on the sandwich ingredients rather than worrying about the oven. In fact, weak object cues might potentially be automatically derived from the objects that the user touches in performing the task. Conceptual and mixed cues might be applied when the user sees that the robot doesn't understand something, or to pre-clarify a complex concept expected to cause confusion: why can salami go on a sandwich but not the raw hamburger next to it in the fridge?

A meteorologist training a severe weather prediction system might convey to the learner her intuition that this part of the storm is particularly important, or instruct it to focus on windspeed early on when trying to determine what causes a tornado.

The synthetic data generator of Appendix A supports automatically adding object or mixed cues, and object can also be automatically added to the dynamics simulation data discussed in Section 3.2.2. In our real-world data collection setup (Section 5.3), we have added a simple interface where an operator presses buttons to apply mixed or object cues during the data collection process. In future environments, however, one could easily imagine running voice recognition software to accept cues, using either a featured context-free grammar (Bird, Klein, & Loper, 2009, Chapter 9) or learning-based techniques (e.g. Kuhlmann et al., 2004) to convert from the string of words spoken to a cuing specification.

Conceptual cues are currently based on $n$-ary relations (in general, consisting of an entire SMRF tree) which have already been learned or perhaps hand-coded. In future work, this vocabulary need not be predefined, in which case these cues become part of a symbol grounding problem for the learner. Techniques

used for this problem in the past (e.g. Dominey & Boucher, 2005; Roy, 1999, 2002; Siskind, 2001; Y. Xu & Kemp, 2010) might be applicable, though the interactions between learning what a cue represents and using it to learn other concepts become complex. Such work will have to wait until the SMRF framework as a whole is more mature.

## 4.3 Implementation in SMRF

### 4.3.1 Object Cues

Object cues, at their core, tell the learner to focus on some objects and avoid others. This is in many ways analogous to the information provided by the Diverse Density calculation of Section 3.1.2, which biases the search for a volume in attribute space towards some areas and away from others. We therefore incorporate object cues into the $h$ initialization process by integrating their information into DD.

In normal DD, probability kernels are placed around each instantiation sequence from the training examples, with weight corresponding to the label of the sequence's graph. We now add kernels around sequences corresponding to cued objects. Specifically, we let object cues have a weight between -1 and 1, where 0 means no cue, 1 means a strongly positive cue, and -1 means a strongly negative cue; the cue weight of an instantiation sequence is defined as the average of its items' cue weights.

We then add a factor for each cued instantiation sequence to the calculation of $\widehat{DD}$ (Equation 2):

$$\prod_{w_I>0} \left(1 - w_I + w_I \Pr\left(I \in c_t\right)\right) \prod_{w_I<0} \left(1 - |w_I| \Pr\left(I \in c_t\right)\right) \tag{5}$$

where $w_I$ is the weight of instantiation sequence $I$. The effect of (5) is that positive cues decrease the evaluation of points far away from the cue location, and negative cues decrease the evaluation near the cue location.

This will push the initialization process towards positively-cued areas of the attribute space and away from negatively-cued areas. Ideally, this will help push the search towards questions about important instantiation sequences early on, which should help quickly focus the search, especially in situations with many distractors.

### 4.3.2 Conceptual Cues

Conceptual cues define subtrees which are considered interesting in the current problem. It is natural, then, to consider simply splicing the cue tree from our vocabulary onto the tree being learned. The question models of the cue tree are then re-optimized, to account for variation between the dataset on which the cues were learned and this data.

We consider this as a possible expansion in the tree growth phase; for mixed cues, we check this expansion only when the cued objects are consistent with the objects to which they would be mapped in the expanded tree. (If we've cued a specific object as being "blue," we only consider this special cue expansion when that object would actually be sorted to the relevant portion of the tree.)

These kinds of expansions do not help us if, as in the version of the SMRF learning algorithm described in Section 3, we always consider all possible expansions to a tree. Instead, we now consider expansions in "chunks," which are ordered according to how well they are expected to perform. The goodness heuristic is currently based on similarity to cues, so that standard expansions have their scores boosted according to

how much they have in common with a cued question. (If a given object has "blue" cued, then absolute color questions involving that object get a ranking boost, while relative color questions get a somewhat smaller boost, and so on.) Specifically, an expansion's goodness score is increased based on its similarity in mapping function and in mapping parameter / attribute template to a cue.

Once a particular chunk has been fully evaluated, if the best-performing one obtained an improvement beyond our statistical significance threshold, we accept it. Otherwise, we continue on to the next chunk of expansions.

Conceptual cues combined with chunking can potentially result in dramatic decreases in runtime, as irrelevant expansions no longer need to be considered. (They do carry with them the danger of ignoring a potentially useful variable which was not cued, however. Perhaps including a few expansions not expected to do well in each chunk would sometimes allow for such discoveries.)

The inclusion of multi-question cues also expands the kind of steps that the algorithm can take through its search space, as normally only one question at a time would ever be considered. Subconcepts which are difficult to learn can simply be included directly in the learned tree.

#### 4.3.3 Mixed Cues

Mixed cues can be implemented essentially through the combination of an object cue and a conceptual cue. In addition, though, we can check in the learning process that the cued objects will actually filter down to a given node before we consider that cue at the leaf in question. (If, at a given leaf, we know that the object cued as "green" does not even reach the leaf, we can simply ignore that cue.)

### 4.4 Experiments

#### 4.4.1 Object Cues in Action Outcome Prediction

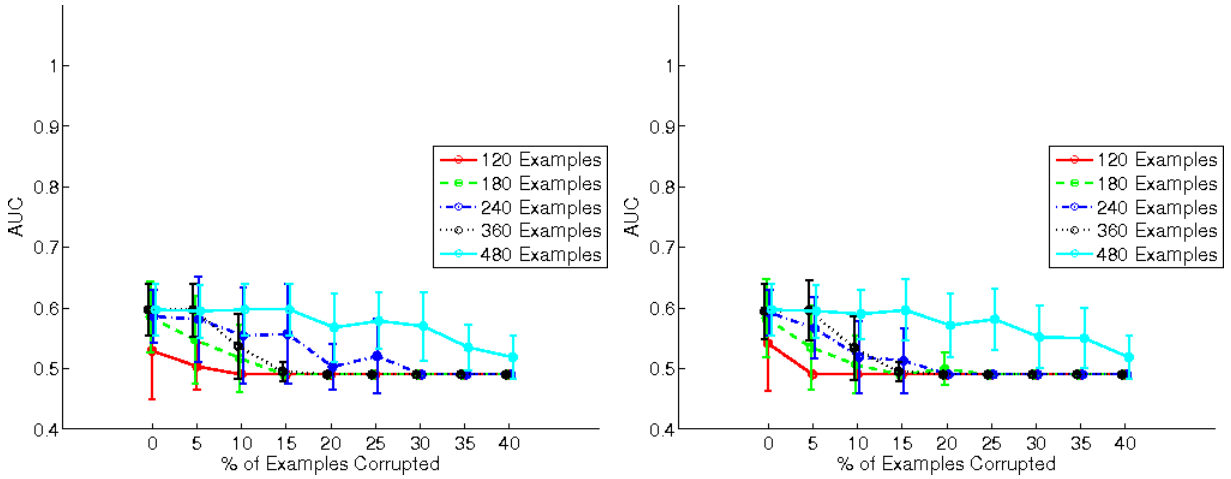
The block-dropping experiment of Section 3.2.2 lends itself naturally to one kind of cue in particular: an object cue on the dropped object. There is no doubt that particular block will play an important role in our determining whether a state will end up with the dropped block on the ground, and so we might naturally incorporate that information into the information given to the model.

We added object cues of weight 1 to the dropped block in each training example. The resulting performance is shown in Figure 12b; the uncued results of Figure 11a are reproduced as Figure 12a for ease of reference. Note that the performances are essentially identical; if anything, the cued version performs slightly worse. (See the next section for discussion.)

It seems, then, that the learning algorithm was already able to identify the most important objects for this simple task; we would need a class where finding the appropriate objects to ask questions about is more difficult to see the effects of object cues on performance.

#### 4.4.2 Mixed Cues in Synthetic Multi-Object Structures

We now evaluate the performance of mixed cues on the blue above green dataset. This data was generated with mixed cues applying to each of the concepts being generated (c.f. Section A.2), so that there are mixed cues for "green," "blue," and "above" on each positive training example.




(a) Uncued results.

(b) Cued results.

Figure 12: The mean and std AUCs for the block-dropping dataset, run (a) without cues and (b) with an object cue on the dropped object. Note the essentially identical performance.

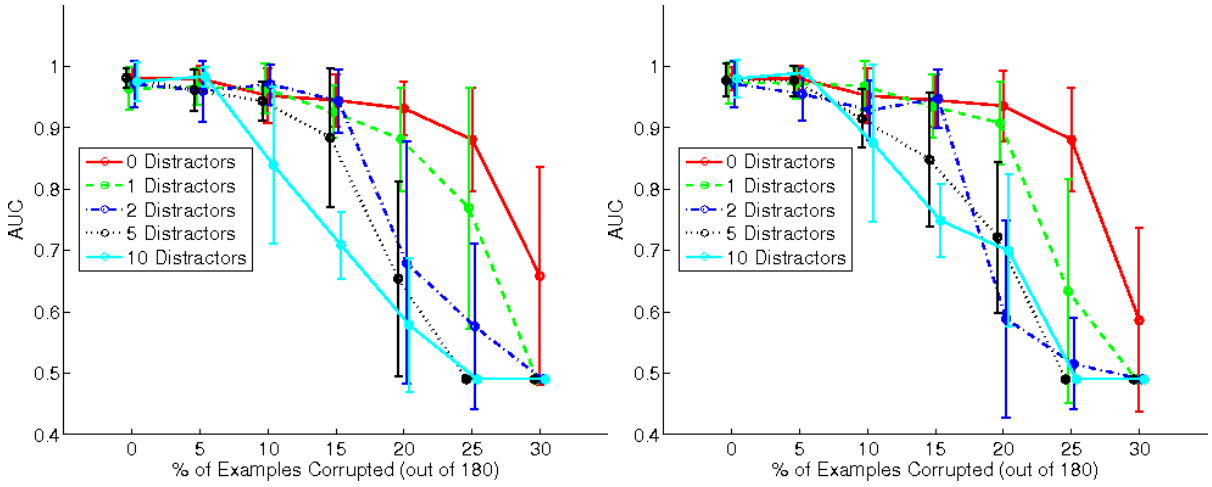
We first consider the heuristic score for expansions, to test that the order output is reasonable — if it were not, chunking would not make sense in the first place. The expansion order for the first expansion of a stub tree in this fully cued situation is shown in Table 1a, and that for a tree following the expansion "green" in Table 1b. The ordering seems reasonable, although the already-used "green" expansion and related color expansions appear to rank too highly in Table 1b — the logic that is supposed to prevent that did not work in this case.

We now turn to the performance results of adding cues to the blue above green problem. Figure 13 presents AUCs for the various cuing options in the presence of corruption; Figure 14 presents the amount of time taken until the learning algorithm concludes it has finished the process.[10]

We can see that on this problem, the addition of cues with no chunking causes essentially no change in performance. Adding chunking has only a very slight negative effect on AUC, but speeds up the learning process considerably (approximately halving the time for most experiments).

Unfortunately, object cues seem to have a substantially negative effect on performance in this instance. This is because both green and blue objects were cued with a weight of 1, causing extremely low values for DD outside of their respective areas of any attribute space – which means that values over the entire space were pushed down essentially to zero, and so DD became much less informative. Further work should experiment with using smaller cue weights or cuing only a subset of the objects present. Future versions of the cuing system should also institute safeguards against overzealous cuing ruining the DD calculation; perhaps finding one $h$ initialization without using the cue factor of (5) would help.

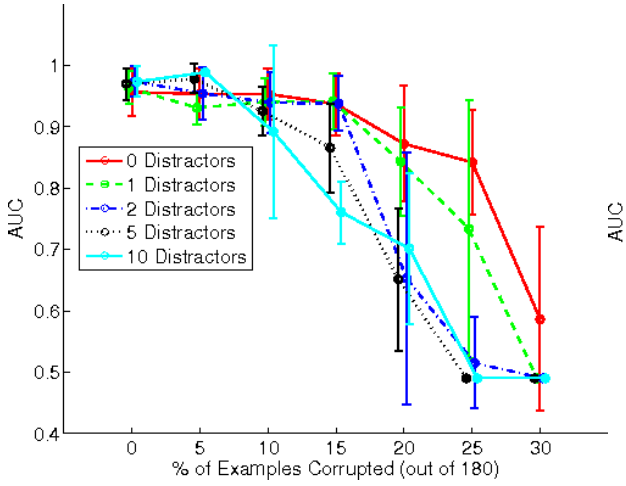
[10]These experiments was run on machines with AMD Opteron processors, one core per trial, with clock speeds ranging from 1.8 to 2.8 GHz; the different folds of cross-validation were distributed essentially uniformly across the machines, and so differences in clock speed should not have much effect on the results shown here.



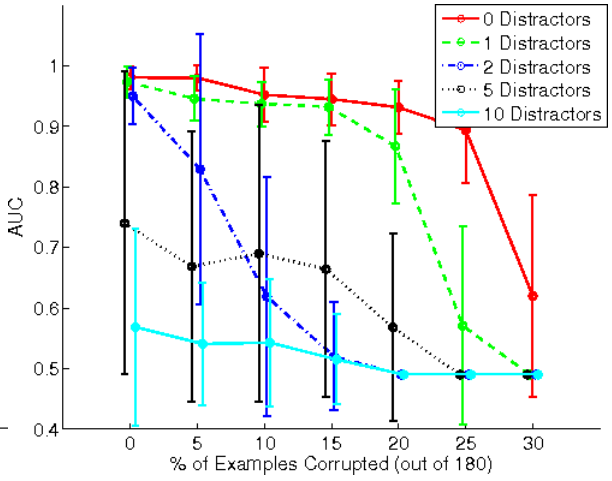

(a) Without cues.




(b) With conceptual cues.

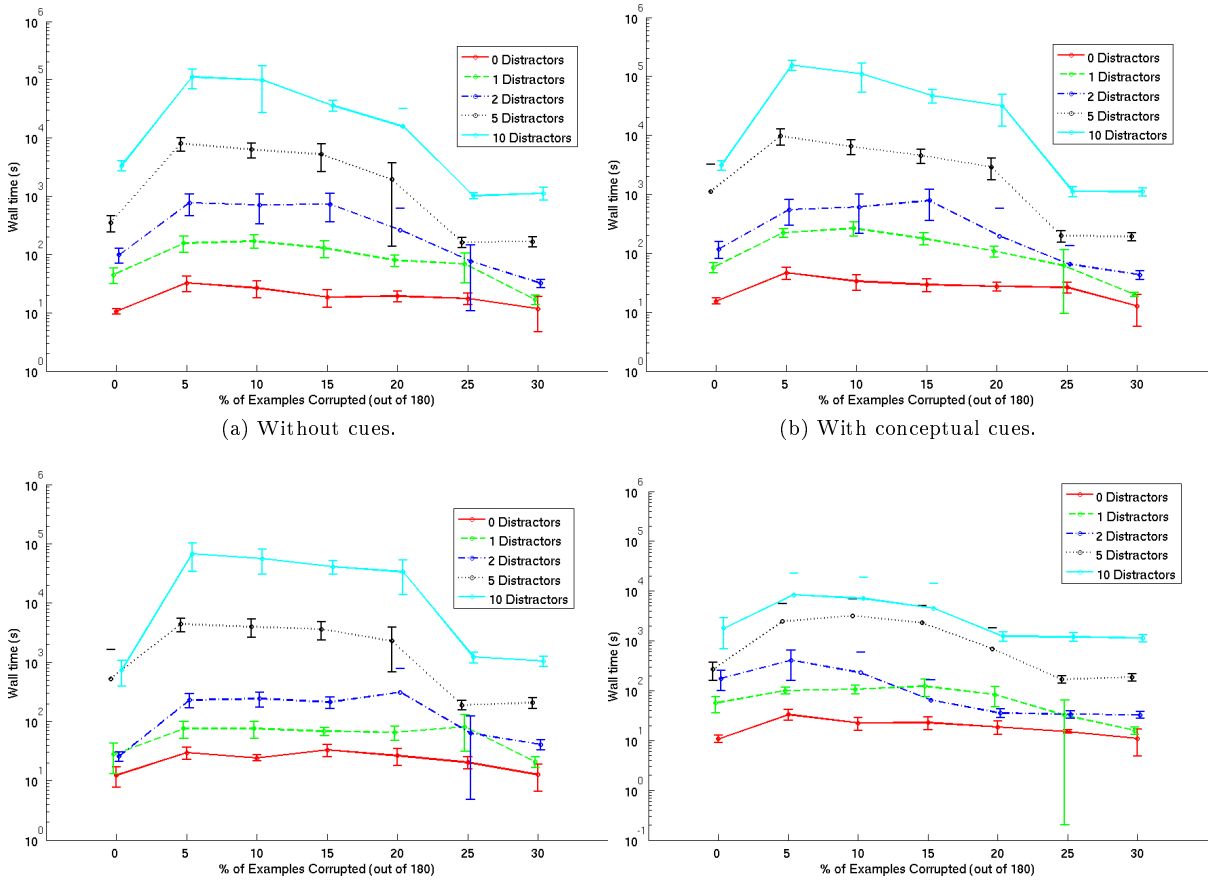



(c) With conceptual cues and chunk size 5.




(d) With object cues.

Figure 13: Cued results for the blue above green problem with corruption.




(a) Without cues.

(b) With conceptual cues.

(c) With conceptual cues and chunk size 5.

(d) With object cues.

Figure 14: Learning time for the cued blue above green problem with corruption. Note the logarithmic scale, which makes the graphs appear more similar.

| # | Expansion | Args | Score |
|---|---|---|---|
| 1 | color | A | 3677 |
| 2 | “blue” | A | 3565 |
| 3 | “green” | A | 3565 |
| 4 | “above” | A, B | 3565 |
| 5 | “above” | B, A | 3565 |
| 6 | relative color | B, A | 2555 |
| 7 | location | A | 2480 |
| 8 | relative location | B, A | 1934 |
| 9 | distance | B, A | 1845 |

(a) For the first expansion.

| # | Expansion | Args | Score |
|---|---|---|---|
| 1 | “blue” | B | 3518 |
| 2 | “green” | B | 3518 |
| 3 | “above” | B, A | 3518 |
| 4 | “above” | B, C | 3518 |
| 5 | “above” | C, B | 3518 |
| 6 | relative color | B, A | 2509 |
| 7 | location | A | 2434 |
| 8 | color | B | 2434 |
| 9 | relative color | C, B | 2135 |
| 10 | relative location | B, A | 1888 |
| 11 | distance | B, A | 1798 |
| 12 | location | B | 1686 |
| 13 | relative location | C, B | 1387 |
| 14 | distance | C, B | 1163 |

(b) Following the first expansion “green.”

Table 1: The ordering of cued expansions in the blue above green problem. The horizontal lines separate expansions which are evaluated in different chunks when running with chunk size five.

# 5 Conclusions and Future Work

We have described a framework for representing a quite general class of concepts in the world, an initial algorithm for learning such representations, and sketched an early method for integrating human domain knowledge into these representations.

All of the work presented here is rather initial, particularly that on cuing, and most aspects of it will probably be changed in a future evolution of the SMRF system. There are many extensions to be made to the framework, much of the learning algorithm is in the process of being reworked (which will necessitate changes to cuing), and the application of the framework to new problem domains and usage modalities will surely bring about changes throughout the system.

Nevertheless, we feel that the framework thus far has shown great promise for the future. It is capable of learning some interesting concepts, and with more work will be able to do more. Its expressive abilities combined with the learned models' being relatively easy for humans to understand should make for a powerful combination in a variety of domains.

## 5.1 Framework Extensions

Possible extensions to the framework are numerous. One particularly important one is the ability to use SMRF trees for regression, rather than just classification, so that the trees could (for example) predict where a tornado is likely to touch down rather than just whether a storm will produce a tornado. Work is underway to implement this; in broad terms, leaf nodes will gain standard regression models, and the tree will be a way to decide which model is most applicable. It thus serves as a categorization layer prior to making other decisions, an approach with support from cognitive science and one which has seen success in many areas (e.g. M. Xu, Watanachaturaporn, Varshney, & Arora, 2005; Lee, Walker, Meeden, & Marshall, 2009).

The other major hurdle to overcome is “meta-instantiation”: the ability to treat a group of objects

as a single meta-object for the purposes of asking certain types of questions. Spatiotemporal Relational Probability Trees and their random forest extensions (McGovern et al., 2008, 2010) do this to a certain degree by supporting conjunctive questions, but it would be desirable to create more flexible categories of grouping in SMRF with richer attributes. The instance-oriented nature of SMRF should also make these groupings more useful by making these objects accessible lower in the tree. One could imagine a significant role for trainer cues to play in defining what kind of meta-objects to consider.

Meta-instantiation could also be used to deal with universally quantified predicates (e.g. "all blocks are yellow," "there is no purple block near a green block"), which the current version of SMRF cannot handle.

The framework itself will also need to support temporal concepts, probably by adding lifetime attributes to objects and then a set of mapping functions that can deal with relative timing. Categorical data is another eventual necessity.

## 5.2 New Learning Approach

As mentioned previously, there are some problems with the learning algorithm presented in Section 3. It would be desirable to use only one metric throughout the learning process (which would help guarantee convergence) and to support disjunctive concepts (the algorithm of Section 3 generally does not work for those purposes, as it tries to capture all of the high-$h$ points in the EM process). Alternative learning approaches might also prove to be faster.

An approach to learning which does not use the concept of $h$ at all is under development. One approach, which proved much slower than the approach of Section 3, used the $R$-algorithm of Shor (1985) to obtain a tree through nonlinear programming. Another approach is currently under development which avoids $h$s by considering only the instantiation sequence from each bag with the highest model evaluation in fitting the question pdfs. This approach allows the algorithm to handle multimodal distributions, and has seen good results in initial testing.

## 5.3 Learning from Demonstration

One area in which future versions of SMRF may do well is the robotics-oriented idea of Learning from Demonstration (LfD). Under this approach, a human teacher provides examples of the target concept, which are then used as training points for the learner. This expert generally has some knowledge of the domain, but may not be a machine learning expert; he or she might be the end-user of a domestic robot, or a meteorologist setting up a severe weather prediction system. See Argall, Chernova, Veloso, and Browning (2009) for an overview of recent research in the LfD field.

In our block world training data, LfD means that the user builds a block structure using a motion tracking setup partially depicted in Figure 1 (page 3). The blocks are initially separated, and one example state is created after each time a block has been completely moved. In each of several trials, one final state is marked as the positive example, while the previous states are entered as negative examples.

At this point, the process requires a fair amount of tedious manual action, and the trainer must be careful to perform the process in the right way. It does provide early indications of how a future, more user-friendly system might work, however — and more importantly for simple tests of the learning framework, the LfD process produces data with "real-world" noise.

Excelling at LfD-type interactions may take more work than simply providing a more friendly interface for data collection, however. Some initial work has taken place in using the trainer for active learning: the user provides some initial training data, SMRF learns as much as it can about the data, and then the example generator of Section A.1 is used to create some examples consistent with the learned tree, which the user identifies as satisfying or not satisfying the target concept.

Work by Thomaz (2006) suggests that a truly successful socially-guided machine learner does more than just use the user as a data source: it provides the user with a natural teaching experience and allows for various kinds of feedback into its operation. Future work on integrating human knowledge into SMRF might profitably consider in more depth the kinds of interactions that a trainer can and wants to make in the learning process, and determine how those interactions might best be incorporated into the algorithm in a way that works for both the human and the algorithm to maximize the system's overall productivity.

## Acknowledgements

I joined the SMRF project after the framework (Section 2) and an early version of the learning algorithm (Section 3) had been created at the University of Oklahoma, primarily by Matthew Bodenhamer and Dr. Andrew Fagg, with assistance from Sam Bleckley, Daniel Fennelly, and Dr. Amy McGovern.

During my REU project, I worked closely with Matt in making substantial improvements to the algorithm and its implementation, fixing many bugs and creating the data generator of Appendix A, among other things, as well as doing the work of Section 4. Dr. Fagg advised us through the process, and Thomas Palmer has since become heavily involved in SMRF as well; many of the ideas described in this thesis are his. I have continued to work on some of the ideas discussed in Section 5.2, among others, in the months since the REU has ended.

Lisa Meeden and Rich Wicentowski also assisted in the preparation of this document.

This material is based upon work supported by the National Science Foundation under Grant No. IIS/REU/0755462.

## References


Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 469–483.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: Analyzing text with the Natural Language Toolkit*. O'Reilly Media.

Bodenhamer, M., Bleckley, S., Fennelly, D., Fagg, A. H., & McGovern, A. (2009). Spatio-temporal multidimensional relational framework trees. In *2009 ieee international conference on data mining workshops* (pp. 564–570). Miami, FL, USA: IEEE.

Dietterich, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, *89*(1-2), 31–71.

Dominey, P. F., & Boucher, J.-D. (2005). Learning to talk about events from narrated video in a construction grammar framework. *Artificial Intelligence*, *167*(1-2), 31–61.

Friedman, J. H. (1977). A recursive partitioning decision rule for nonparametric classification. *IEEE Transactions on Computers*, *C-26*(4), 404–408.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*. Cambridge, Massachusetts: MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

Huelsenbeck, J. P., & Crandall, K. A. (1997). Phylogeny estimation and hypothesis testing using maximum likelihood. *Annual Review of Ecology and Systematics*, *28*, 437–466.

Kemp, C., & Jern, A. (2009). Abstraction and relational learning. *Advances in Neural Information Processing Systems*, *22*.

Knox, W. B., & Stone, P. (2010). Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of 9th int. conf. on autonomous agents and multiagent systems (aamas 2010)*. Toronto.

Knox, W. B., & Stone, P. (2011). Reinforcement learning with human feedback in mountain car. In *Aaai spring 2011 symposium on bridging the gaps in human-agent collaboration.*

Kuhlmann, G., Stone, P., Mooney, R., & Shavlik, J. (2004). Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *Proceedings of the aaai-04 workshop on supervisory control of learning and adaptive systems.*

Lee, R., Walker, R., Meeden, L., & Marshall, J. (2009). Category-based intrinsic motivation. In *Proceedings of the ninth international conference on epigenetic robotics.*

Maclin, R., Shavlik, J., Torrey, L., Walker, T., & Wild, E. (2005). Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In A. Cohn (Ed.), *Proceedings of the 20th national conference on artificial intelligence* (Vol. 2, pp. 819–824). AAAI Press.

Maron, O. (1998). *Learning from ambiguity*. Phd thesis, Massachusetts Institute of Technology.

Maron, O., & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Advances in neural information processing systems* (Vol. 10, pp. 570–576). Cambridge, Massachusetts: MIT Press.

McGovern, A., Hiers, N. C., Collier, M., Gagne, II, D. J., & Brown, R. A. (2008). Spatiotemporal relational probability trees: An introduction. In *Proceedings of the international conference on data mining* (pp. 935–940). Pisa, Italy.

McGovern, A., Supinie, T., Gagne II, D. J., Collier, M., Brown, R., Basara, J., et al. (2010). Understanding severe weather processes through spatiotemporal relational random forests. In *Proceedings of the nasa conference on intelligent data understanding: Cidu 2010* (pp. 1–15).

Neville, J., Jensen, D., Friedland, L., & Hay, M. (2003). Learning relational probability trees. In *Proceedings of the ninth acm sigkdd international conference on knowledge discovery and data mining* (pp. 625–630). Washington, D.C.: ACM.

Provost, F., & Domingos, P. (2003). Tree induction for probability-based ranking. *Machine Learning*, *52*(3), 199–215.

Roy, D. K. (1999). *Learning words from sights and sounds: A computational model.* Phd thesis, Massachusetts Institute of Technology.

Roy, D. K. (2002). Learning visually grounded words and syntax for a scene description task. *Computer Speech & Language*, *16*(3-4), 353–385.

Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (2 ed.). Prentice Hall.

Shor, N. Z. (1985). *Minimization methods for non-differentiable functions.* New York: Springer-Verlag.

Siskind, J. M. (2001). Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research*, *15*, 31–90.

Thomaz, A. L. (2006). *Socially guided machine learning.* Phd thesis, Massachussets Institute of Technology.

Walker, T., O'Reilly, C., Kunapuli, G., Natarajan, S., Maclin, R., Page, D., et al. (2010). Automating the ILP setup task: Converting user advice about specific examples into general background knowledge. In *Proceedings of the 20th international conference on inductive logic programming* (pp. 1–16).

Xu, M., Watanachaturaporn, P., Varshney, P., & Arora, M. (2005). Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment*, *97*(3), 322–336.

Xu, Y., & Kemp, C. (2010). Constructing spatial concepts from universal primitives. In *32nd annual conference of the cognitive science society.*

Zhang, Q., & Goldman, S. A. (2001). EM-DD: An improved multiple-instance learning technique. In *Advances in neural information processing systems* (Vol. 14, pp. 1073–1080). MIT Press.

# A Synthetic Data Generation

Much of the data for experiments used in this thesis was obtained through a synthetic data generator. Data is created by defining a set of SMRF models applying to named objects, the conjunction of which defines the target concept. (Work to extend the generator to disjunctive concepts is ongoing.)

Broadly speaking, the generator defines SMRF trees based on the given concepts to provide a range of types of negative and positive examples, then generates examples that filter down to a certain leaf in each of those trees. We first discuss the system which generates examples for a given leaf of a tree.

## A.1 Brownian Example Generator

The example generator takes as input a SMRF tree, a desired leaf, and at least one instantiation sequence which sorts to the desired leaf. It then tweaks the attributes of each item, backtracking if a given step causes the instantiation sequence to sort to a different leaf; the final result is a different instantiation sequence which still reaches the same leaf of the tree.

For certain kinds of models, particularly those performing more complex operations on multiple objects, this kind of generation is not ideal. It can only step in one attribute at a time, and so to, say, move the entire coordinate frame of a relative orientation question requires a good deal of luck. More complex kinds of steps through instantiation sequence space would be helpful in these situations.

The generator is called Brownian by analogy with Brownian motion. Perhaps more precisely, it comprises a discrete-time random walk through the (continuous) space of instantiation sequences of a given form and length, except that steps are made one-by-one in each dimension, and each dimension is bounded.

The example generator also sees use in "interactive" learning problems, mentioned in Section 5.3 but not discussed at length in this thesis.

## A.2 Data Generation Algorithm

Generating synthetic data requires three pieces of input:

1. A list of concepts, which associate instantiation sequences of named objects to a SMRF mapping function (see Section 2.1.1) which they must satisfy. In the blue above green problem, named items 'b' and 'g' are associated to color models like that of Figure 3, and the sequence ('b', 'g') is associated to a relative location model representing "above."
2. A set of positive graphs from which to generate examples.
3. Extra attribute values to base negative examples off of; in blue above green, these include a "red" color model and locations off to the side.

The generation process then proceeds by deriving a set of base items, which is the set of all possible combinations of attribute values seen in the positive graphs and extra values. It then determines which lists of these items can satisfy each question: which of the generated items satisfy the "blue" model, which pairs of items satisfy the "above" model, and so on.

From there, the generator samples a large number of question combinations for negative graphs. These consist of any number of questions which should be true, plus exactly one which should be false; for blue above green, these include "blue but no green," "blue and green but not above," "not above." Note that since

objects are named, "blue and green but not above" would include examples with a green object above a blue object.

From each of those combinations, the generator uses a standard constraint satisfaction algorithm (Russell & Norvig, 2002, Chapter 5) to find a small number of base instantiation sequence examples for the desired leaf, then generates the required number of outputs using the example generator.

Positive examples are generated simply by running the example generator on the positive input graphs.