# An Example of Learning in Knowledge-Directed Vision

Bruce A. Draper Allen R. Hanson
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA., USA. 01003*

## 1 Introduction

The goal of image understanding systems is typically the identification of objects in visual imagery and the establishment of the three-dimensional relationships among the objects and the viewer. It is a generally accepted premise that, in many domains, the timely and appropriate use of relevant knowledge can substantially reduce the combinatorially explosive search encountered in establishing 'instance-of' relationships between image data and its interpretation(s).

Because of the variety and scope of knowledge pertinent to vision, the acquisition of both object models and interpretation strategies remains a major outstanding problem in model-based image understanding. While many vision algorithms at the low and intermediate levels are available, successful use of knowledge in image understanding requires a careful hand-crafting of the knowledge base. Typically this requires specifying, for each object class, both the description of the generic object as well as one or more recognition (control) strategies for instantiating instances of the object to image data.

The success of many knowledge-based image understanding systems can be traced to a "small world" assumption, in which the number of objects in the domain are few, the constraints on their descriptions are tight, and a complete world model is at least a possibility. Consequently, special purpose systems are able to define, structure, and apply relevant task knowledge effectively. However, as the scope of a system broadens towards a domain-independent, general-purpose system, an unfortunate chain of events occurs: the size of the knowledge base increases, constraints on the object descriptions become looser to account for wider variability, the system can make fewer assumptions about the types of image descriptions necessary for matching, and the complexity of matching increases substantially.

There are really two issues being discussed here, relating to the structure of object and control knowledge in vision systems, and the acquisition of this knowledge. The next section briefly describes the knowledge component of the VISIONS image understanding system ([6, 7]) known as the schema system ([3]), from the point of view of knowledge structuring and control. Subsequent sections discuss the role of learning in the automatic acquisition of portions of the knowledge base. It is our contention that learning techniques must be embedded in vision systems of the future in order to reduce or eliminate the knowledge engineering aspects of system construction.

*This work was supported in part by NSF CII grant CDA-8922572 and by U.S. Army ETL under grant DACA76-89-C-0017

## 2 Summary of the VISIONS Schema System

The success of systems based on the "small world" assumption has led to the adoption of a primary design philosophy for the knowledge component of an image understanding system: both knowledge and computation should be partitioned at a coarse-grained semantic level. In the VISIONS system, this coarse-grained knowledge is encapsulated in a schema. Each schema is specialized to a single object class. This encapsulation permits each schema to be an "expert" in the recognition of instances of the object class and allows the wide range of control strategies necessary for different objects to be represented in a natural way.

A schema instance is invoked for each object class hypothesized to be in the image data. These instances execute independent (potentially concurrent) processes called recognition strategies and communicate asynchronously through a global blackboard. The control component of each schema directs the application of general purpose procedures, called knowledge sources, to gather the "right kind" of support for (or against) its hypothesis. Competition and cooperation among the schema instances results in the combination of multiple, independent "object experts" into a large scale system which constructs internally consistent interpretations.

### 2.1 Components of the Schema System

The schema system consists of five basic components: the schemas and schema hierarchy, the blackboard, the knowledge sources, the interpretation (control) strategies, and mechanisms for evidence representation and combination. Each of these is discussed very briefly in the following sections; more detail may be found in [3].

#### 2.1.1 Schemas and the Schema Hierarchy

The schema system partitions both knowledge and computation in terms of natural object classes for a given domain. Schemas reside in class and part/subpart hierarchies; each class of objects and object parts has a corresponding schema which stores all object and control knowledge specific to that class. Knowledge about expected object contexts and relationships to other objects is represented in the system by extending the concept of an object to include contextual or scene configurations; as objects, these entities also have schemas. A subcontext or "sub-scene" is like an object part; it is related to its parent scene or context in predicatable ways.

#### 2.1.2 Knowledge Sources

Knowledge sources are general-purpose procedures that generate the levels of abstract image descriptions required in an image understanding. Knowledge sources span the gamut of traditional techniques in image processing (e.g. region, line, curve, and surface extraction, feature measurement, etc), through intermediate level processes such as initial object hypothesis generation and grouping operations to generally useful tools and techniques such as graph matching. The compile-time arguments and parameters supplied to a general-purpose knowledge sources as part of the recognition strategy may specialize it for a particular purpose.

KSs typically create, manipulate, and construct abstract symbolic representations of image events stored symbolically in the ISR [2], a database specially designed for image understanding systems. The database supports associative retrieval, spatial relations, multi-level representations, and has been optimized for spatial retrieval. In the current version of the schema system, which has recently been extended to included three-dimensional object representations and three- dimensional interpretation, over 40 KSs are available as basic building blocks.

### 2.1.3 Interpretation Strategies

Interpretation strategies, or simply strategies, are control programs that run within each schema. Strategies procedurally encode knowledge about which knowledge sources to apply and in what order to apply them. In order to make maximal use of parallelism, schemas may have multiple concurrent strategies. These strategies may correspond to different methods for recognizing an object or to different conditions under which recognition must take place. Schemas can also contain strategies for different subtasks, such as initial hypothesis generation and hypothesis verification, as well as for managing the internal bookkeeping details of the schema, such as updating the global blackboard when necessary and detecting and resolving conflicts related to the hypothesis.

Each schema instance acquires information pertinent to the hypothesis it is pursuing. Some of this information is generic, to the extent that its semantics are not object dependent. For example, the degree of confidence in a hypothesis, as well as its (2D) image location and (3D) world location, is generic information. Every object hypothesis has a confidence level and an image location, and most have a meaningful 3D location. The generic information about an object hypothesis is recorded in a global hypothesis.

Most of the information acquired by a schema instance, on the other hand, is object specific. Information about how well an image region matches an expected color, for example, is non-generic since its importance depends on the object model. A color match may be important for finding trees, but less so for recognizing automobiles. For this reason, all of the information about which KSs support a particular hypothesis and which do not is considered private to the schema instance, and is not included in the global hypothesis.

### 2.1.4 Blackboard Communication

The schema system is built around a global blackboard. The global hypotheses written to the blackboard represent the image interpretation as it evolves. Schemas communicate with each other by writing to and reading from the blackboard, dynamically exchanging information about their respective hypotheses. Although the blackboard is divided into sections corresponding to the object classes (rather than processing levels, as in other systems [12]), schemas may read and write freely over the entire blackboard. The division into sections gives some assurance that a schema will not have to search through a large number of irrelevant messages. At the same time, each schema instance maintains its own local blackboard for recording private information.

The distinction between the global and local blackboards was motivated both by computational and knowledge engineering concerns. Computationally, most of the information generated by an interpretation strategy concerns which KSs have been run, what each KS returned, etc. While this information is crucially important within the schema instance for making dynamic control decisions, it is of little importance to other schema instances. If the strategies associated with multiple concurrent schema instances continually dump this information to the global blackboard and then read it back again, the blackboard quickly becomes a computational bottleneck. The local blackboards alleviate this problem by reducing the message traffic on the global blackboard.

From a knowledge engineering viewpoint, the distinction between the global and local blackboards promotes modularity. By allowing only the strict "global hypothesis" protocol to be exchanged between schemas, the schema system encourages modularity. Each schema can maintain local information in an idiosyncratic manner on its local blackboard, allowing the schema designer the freedom of any appropriate knowledge representation and control style. At the same time, because schemas communicate with each other only through global hypotheses, the designer of a new schema is assured of a smooth join to the remainder of the system.

#### 2.1.5 Evidence Accumulation

The current version of the schema system takes a particularly simple view of evidence representation and combination. Confidence values lie along a coarse, five point ordinal scale: 'no evidence', 'slim-evidence', 'partial-support', 'belief', and 'strong-belief'. When combining evidence, a heuristic mechanism is used that involves the specification of key pieces of evidence that are required to post an object hypothesis with a given confidence to the global blackboard. Subsets of secondary evidence are used to raise or lower these confidences. Specifications of these subsets, and the effect their confidence has on the overall confidence, is part of the knowledge engineering effort involved in constructing a schema. Although this method of evidence representation and accumulation may lack considerably from a theoretical point of view, it has worked surprisingly well in interpretation experiments on images of New England house and road scenes [3].

### 2.2 Knowledge Engineering in the Schema System

Schemas are assembled by specifying (1) the appropriate set of knowledge sources to be used, (2) a set of strategies which conditionally sequence their application, and (3) a function to translate internal evidence into a confidence in the global hypothesis. One of the main impediments to wide scale experimentation with the schema system has been the time and energy required to design a schema. Schema construction can be viewed as an exercise in experimental engineering, in which prototype schemas are developed using existing system resources. These schemas must then be tested on a representative set of objects/images, failures noted and analyzed, and the schemas re- engineered to account for the failures. In many cases, the descriptive information provided by the knowledge sources may be inadequate. In this case, new knowledge sources must be developed and tested (often a major research effort in its own right), integrated into the system, and the schemas re-engineered to make use of the new information.

The problem of knowledge base construction has been a focus of research for several years. In artificial intelligence, researchers have focused on how to extract knowledge from experts, a scenario which does not apply to computer vision. Vision researchers have concentrated instead on how knowledge bases are specified. By restricting the message types written to the global blackboard, the schema system enforces schema modularity in an attempt to make them easier to declare and improve. The SPAM project at CMU went even farther, developing a high-level language for describing objects ([11]). Work in Japan has involved both automatic programming efforts and higher-level languages for specifying image operations ([10]).

## 3 Learning in Knowledge-Directed Vision

For the last two years we have taken a different approach to knowledge base development. Instead of making the knowledge base easier to program, we have decided to take the programmer out of the loop. Our goal is a knowledge-directed vision system that learns its own interpretation strategies.

As a first step toward achieving this goal we have designed the Schema Learning System (SLS; [4]), as shown in Figure 1. SLS's task is to learn interpretation strategies for the different object classes in a domain. In particular, its task is to learn a strategy that minimizes the cost of object recognition, subject to accuracy constraints supplied by the user. For example, a user might ask for a strategy for recognizing the (3D) position and pose of a building, accurate to within 5%. SLS would then learn a strategy that satisfied this goal by experimenting with training images. Once learned, the strategy is available any time the user needs to locate a building.

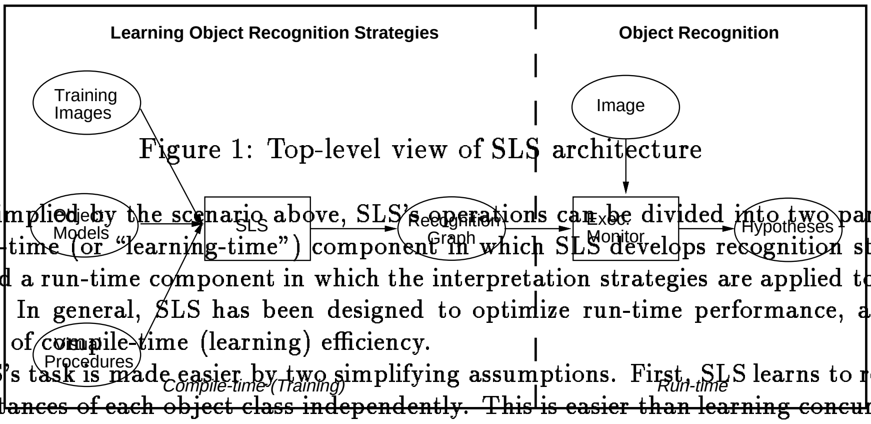

Figure 1: Top-level view of SLS architecture

As implied by the scenario above, SLS's operations can be divided into two parts: a compile-time (or "learning-time") component in which SLS develops recognition strategies, and a run-time component in which the interpretation strategies are applied to new images. In general, SLS has been designed to optimize run-time performance, at the expense of compile-time (learning) efficiency.

SLS's task is made easier by two simplifying assumptions. First, SLS learns to recognize instances of each object class independently. This is easier than learning concurrent, cooperating strategies. Second, SLS is given a set of parameterized knowledge sources from which to build its recognition strategies. Thus we are not asking SLS to learn new knowledge sources or 3D object models. Instead, we are asking it to learn control and evidence combination. It must decide which knowledge sources to execute when, and how to combine the evidence from multiple knowledge sources into a single confidence value. Implicit in this task statement is the possibility that the object model may not be completely accurate, and that some of the knowledge sources may be misleading or irrelevant.

## 3.1 Modeling the Interpretation Process

SLS, like the schema system before it, adopts the blackboard model of interpretation. In other words, interpretation is viewed as a process of applying knowledge sources to hypotheses. Hypotheses are proposed statements about the image or its interpretation, whose type is determined by their *level of abstraction.* Common levels of abstraction for computer vision include: image, region, (2D) image line segment, line segment group, (3D) world line segment, (3D) orientation vector, surface patch, face, volume, object and context. An "interpretation" is a set of believed hypotheses at the level of abstraction requested by the user. Knowledge sources are procedures from the image understanding literature (e.g. region segmentation, line extraction or vanishing point analysis) that can be applied to one or more hypotheses.

SLS, however, refines the blackboard model of interpretation by constraining knowledge sources to fall into one of two classes. *Generation knowledge sources* (GKSs) create new hypotheses. For example, region segmentation is a GKS, since it creates new hypotheses (regions) when applied to an old hypothesis (the image); a stereo line matching algorithm, which produces a (3D) world line segment from two (2D) image line segments, is another GKS. *Verification knowledge sources* (VKSs), on the other hand, return discrete evidence values about the hypotheses they are applied to. An example of a VKS is a pattern matching algorithm that determines if the color or texture of an image region matches the expected color or texture of the object. In general, any routine which measures features of hypotheses can be converted into a VKS by discretizing its results.

## 3.2 Recognition Graphs

Interpretation strategies are represented in SLS as generalized multi-level decision trees called *recognition graphs* that direct both hypothesis formation and hypothesis verification, as shown in Figure 2. The premise behind the formalism is that object recognition is a series of small verification tasks interleaved with representational transformations. Recognition begins with trying to verify hypotheses at a low level of abstraction, separating to the extent possible hypotheses that are reliable from those that are not. Verified hypotheses (or at least, hypotheses that have not been rejected) are then transformed to a higher level of abstraction, where a new verification process takes place. The cycle of verification followed by transformation continues until hypotheses are verified at the goal level of abstraction (as specified by the user), or until all hypotheses have been rejected.

The structure of the recognition graph reflects the verification/transformation cycle. Each level of the recognition graph is a decision tree that controls hypothesis verification at one level of abstraction by invoking VKSs to gather support for or against each hypothesis. When the decision tree determines that a hypothesis is reliable, a GKS transforms it to another level of abstraction, where the process repeats itself.

As defined in the field of operations research, decision trees are a form of state-space representation composed of alternating *choice states* and *chance states.* When searching for a path from the start state to a goal state, an agent is only allowed to choose where to go next from a choice state. If the current state is a chance state the next state is

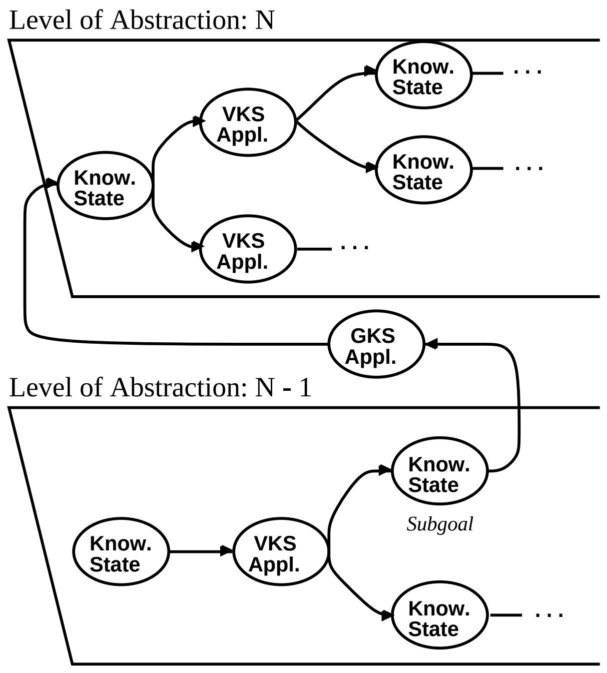

Figure 2: A recognition graph. Levels of the graph are decision trees that verify hypotheses using VKSs. Hypotheses that reach a subgoal are transformed to the next level of abstraction by a GKS.

selected probabilistically[1]. The search process is therefore similar to using a game tree against a probabilistic opponent.

In SLS, the choice states are hypothesis knowledge states as represented by sets of hypothesis feature values. The choice to be made at each knowledge state is which VKS (if any) to execute next. Chance states in the tree represent VKS applications, where the chance is on which value the VKS will return. Hypothesis verification is an alternating cycle in which the control strategy selects which VKS to invoke next (i.e., which feature to compute), and the VKS probabilistically returns a feature value. Thus hypotheses advance from knowledge states to VKS application states and then on to new knowledge states. The cycle continues for each hypothesis until it reaches a subgoal state, indicating that it has been verified and should be transformed to a higher level of abstraction, or a failure state, indicating that the hypothesis is unreliable and should be rejected.

In general, SLS learns in advance what VKS to choose at each knowledge state in order to avoid making run-time control decisions. As a result, when SLS builds a recognition graph it leaves just one option at each choice node. Sometimes, however, the readiness of a VKS to be executed cannot be determined until run-time, in which

[1]Operations research terminology is based on trees rather than spaces, so it refers to choice nodes and chance nodes rather than choice states and chance states, and to leaf nodes and root nodes rather than goal states and start states.

case SLS will leave several options at a choice node, sorted in order of desirability[2]. At run-time the system will choose the highest-ranking VKS that is ready to be executed.

It should be pointed out that recognition graphs can represent a variety of strategies. Bottom-up strategies are represented by having GKS links point "up" the hierarchy, i.e. by having them generate more abstract hypotheses from less abstract ones. Top-down strategies are implemented by having the GKS links point the other way. Most strategies are mixed, in that they have GKS links going both up and down the hierarchy. For example, 2D line segment hypotheses can either be extracted from the image (bottom-up) or predicted by an object pose hypothesis (top-down). Refinement strategies can be represented through seemingly circular links representing iterative processing, as when a perspective KS projects a 3D pose hypothesis, creating 2D line segment hypotheses which are compared to lines extracted from the image and used to generate a more accurate pose hypothesis (which, if verified, can be used to revise the 2D line segment hypotheses...).

## 3.3 The Schema Learning System (SLS)

The Schema Learning System (SLS) constructs interpretation strategies represented as recognition graphs. SLS is given (1) a set of parameterized knowledge sources; (2) a set of user-interpreted training images; and (3) a goal, in terms of a target representation and the required accuracy. It produces a recognition strategy that minimizes the expected cost of achieving the goal.

### 3.3.1 Exploration

SLS learns recognition strategies through a three-step process of exploration, learning from examples, and optimization. The first step, exploration, is algorithmically the least interesting. It exhaustively applies all available knowledge sources (both VKSs and GKSs) to the training images in order to estimate the expected cost of each knowledge source (measured as execution time) and the probability of each VKS result. The exhaustive exploration phase also produces as many correct interpretations as possible with the existing GKSs to serve as examples for the second phase of learning. At this point computational efficiency is unimportant, since the goal is run-time efficiency.

### 3.3.2 Learning from Examples

SLS's second step looks at the correct interpretations produced during exploration and infers from them a scheme for generating good hypotheses while minimizing the number of false hypotheses by tracing back the GKSs employed to produce each good hypotheses. For example, a correct 3D pose hypothesis might be generated by fitting a plane to a set of 3D line segments. If so, the pose hypothesis is dependent on the plane fitting GKS. It is also dependent on whatever GKS created the 3D line segments, and any GKSs needed to create its arguments, etc. The result of tracing back a hypothesis' dependencies is an AND/OR tree like the one shown in Figure 3. 'AND' nodes in the tree result from

[2]This is just one of many complications that arise from multiple-argument knowledge sources. In general, we will describe SLS as if all KSs took just one argument in order to keep the description brief; see [5] for a more complete description.

GKSs that require multiple arguments, such as stereo matching. 'OR' nodes in the tree occur when a hypothesis is redundantly generated by more than one GKS (or a single GKS applied to alternate hypotheses).

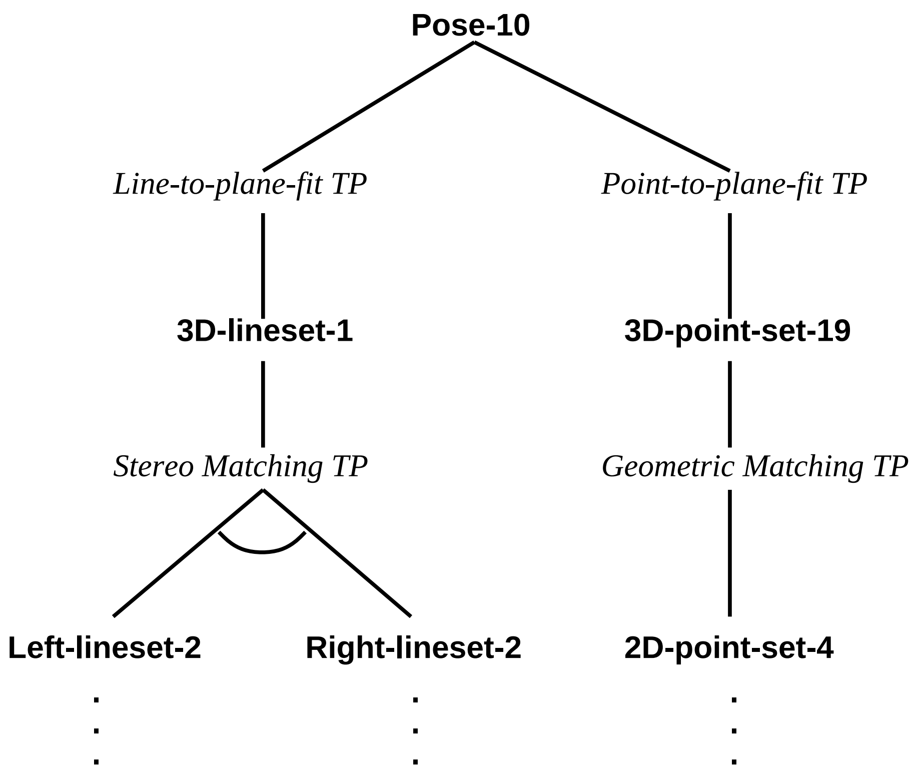

Figure 3: An example of a dependency tree showing the different ways that one correct pose hypothesis can be created during training.

Each dependency tree is viewed as an example of how correct hypotheses are created. The example is generalized by replacing the hypotheses in the tree with their feature vectors. In other words, instead of viewing Figure 3 as showing how pose-10 was created during training, we interpret it as an example showing how poses can be created by a specific GKS (e.g. the GKS for fitting lines to planes) when applied to hypotheses with specific features (in this case, the feature values of 3D-lineset-1).

Since the goal is to learn a strategy that will generate (and later verify) all instances of an object or object class, SLS collects the dependency trees of all the correct hypotheses into a single multi-sample tree by ANDing their root nodes together. By definition, any set of conditioned GKSs (i.e. GKSs with specific feature values as preconditions to the arguments derived from the training set) that satisfies this tree will generate all the correct hypotheses over the training images. However, there is no reason to believe that such a set of GKSs will generate only correct hypotheses; it will generate incorrect ones as well. Therefore, SLS's job in step two is to find a set of (conditioned) GKSs that satisfies the multi-sample dependency tree while minimizing the number of incorrect hypotheses generated.

SLS finds the optimal set of generation knowledge sources (GKSs) by converting the multi-sample dependency tree into disjunctive normal form (DNF) and selecting the conjunctive subterm that generates the fewest incorrect hypotheses. Because of the way the tree was constructed, the GKSs in any subterm are sufficient to generate correct goal-level hypotheses for every object instance in the training set.

The AND/OR dependency tree is converted into DNF by a standard algorithm that first converts its subtrees to DNF and then either merges the subterms (if the root is an OR node) or takes the symbolic cross-product of the subterms (if the root is an AND node). SLS, however, is designed to find just the minimal term of the resulting DNF expression; as a result, any time during the conversion process that a DNF has two subterms one of which is a logical superset of the other, the superset term can be pruned from the expression.

Readers may note that converting an arbitrary AND/OR tree to DNF is an exponentially expensive process: in the worst case, a tree with $N$ literals can produce a DNF expression with $2^N$ subterms. In the case of SLS, the pruning condition reduces the worst-case complexity to $N$ choose $\lfloor \frac{N}{2} \rfloor$, but this is still exponential. The worst-case analysis, however, is largely inappropriate because it corresponds to random data. As long as the samples in the training set are visually similar (and unless all of the knowledge sources produce random results) the worst case will never arise.

### 3.3.3 Optimization

As was stated earlier, recognition graphs interleave verification and transformation, using VKSs to gather evidence to verify or reject hypotheses, and GKSs to transform them to higher levels of abstraction. By building a dependency tree from the training samples, converting it to DNF and picking the minimal subterm (measured by the number of incorrect hypotheses generated), SLS learned which GKSs to include in a strategy that generates correct hypotheses while minimizing the number of false alarms (and presumably cost). Just as important, it learned what evidence to require of a hypothesis before it should be transformed. The subterms of the DNF expression are GKSs constrained to be applied to hypotheses with specific sets of features, and these feature sets are the subgoals of the recognition process.

In the third step of the algorithm, SLS optimizes recognition by building decision trees for each level of abstraction that minimize the expected cost of reaching a subgoal or, conversely, of deciding that a hypothesis cannot satisfy any subgoal and should be rejected. This is achieved at each level by first laying out the graph of all possible sequences of knowledge states and VKS applications and then pruning it to leave just the tree that minimizes the expected cost.

For each level of abstraction, the initial graph layout begins with a start state. VKS applications are added for every VKS that can be applied to a hypothesis in the start state, and these VKS applications lead to new knowledge states, which in turn have more VKS applications attached to them, and so on. The expansion of the graph continues until it reaches either a subgoal knowledge state or a knowledge state that is incompatible with every remaining subgoal (i.e. a failure state).

Once the initial graph has been laid out, SLS begins to prune it by working backwards from the subgoal and failure nodes toward the start state. At each VKS application node it calculates the expected cost of reaching a subgoal or failure node from that particular application node. At each knowledge state, it finds which of the possible VKS applications has the lowest expected cost and removes the other VKSs from the list of candidates (in the event that the optimal VKS might not be executable at run time, it sorts the remaining VKSs in order of least to greatest expected cost rather than removing them.

More formally, we refer to the subgoal states and the failure states at one level of a recognition graph as the *terminal* states for that level. The cost of promoting a hypothesis from knowledge state $n$ to a terminal state is called the Expected Decision Cost (EDC) of state $n$, and the expected cost of reaching a terminal state from state $n$ using VKS $k$ is the Expected Path Cost (EPC) of $n$ and $k$. Since verification KSs return discrete values, we refer to the possible outcomes of a verification KS $k$ as $R(k)$, and the probability of a particular value $e$ being returned as $P(e|k,n), e \in R(k)$.

The EDC's of knowledge states can be calculated starting with the terminal states and working backwards through the recognition graph. Clearly, the EDC of a subgoal or failure state is zero:

$$EDC(n) = 0, \quad n \in \{terminal\ states\}.$$

The expected path cost of reaching a terminal state using a particular VKS is:

$$EPC(n,k) = C(k) + \sum_{e \in R(k)} (P(e|n,k) \times EDC(n \cup e))$$

where $n$ is the knowledge state expressed as a set of feature values, $n \cup e$ is the knowledge state that results from VKS $k$ returning feature value $e$ and $C(k)$ is the estimated cost of applying $k$.

The EDC of a knowledge state, then, is the smallest EPC of the knowledge sources that can be executed at that state:

$$EDC(n) = \min_{k \in KS(n)} (EPC(n,k))$$

where $KS(n)$ is the set of VKSs applicable at node $n$.

The equations above establish a mutually recursive definition of the expected decision cost of a knowledge state. The EDC of a knowledge state is the EPC of the optimal VKS application at the state; the EPC of a VKS application is the expected cost of applying the VKS plus the expected remaining EDC after the VKS has been applied. The recursion bottoms out at terminal nodes, whose EDC is zero. Since every path through the object recognition graph ends at either a subgoal or a failure node, the recursion is well defined. Furthermore, since the EDC of a level's start state estimates the expected cost of verifying a hypothesis at that level of abstraction, the EDCs of all the start states can be combined with estimates of the number of hypotheses generated at each level to estimate the expected run-time of the strategy as a whole.

## 3.4 Preliminary Results

The previous sections give a simplified description of a complex system that has only recently been implemented. Because the system is new, complete and thorough experiments to examine its behavior on real images have not yet been run. Preliminary experiments testing its success both as a knowledge engineering tool and as a machine learning system are underway; in this section we report the results of one such experiment.

The goal of the experiment was to test (1) how long it would take using SLS to develop a strategy for recognizing a building from an approximately known viewpoint and (2) how robust the resulting strategy would prove to be. To this end fourteen images like the one shown if Figure 4 were taken of the Marcus Engineering Building on the UMass campus from a dirt path three to four hundred feet from the building. The pictures were taken so that the image's y-axis would be parallel to gravity (i.e. with zero tilt and roll), however there were significant rotations in pan from one image to the next; as a result, the pose of the building has four free parameters, three locational and one rotational. SLS's goal was to learn a strategy that could identify the pose of the building to within $10^\circ$ rotation, 10% depth (scale) and $1^\circ$ of the correct image angle.

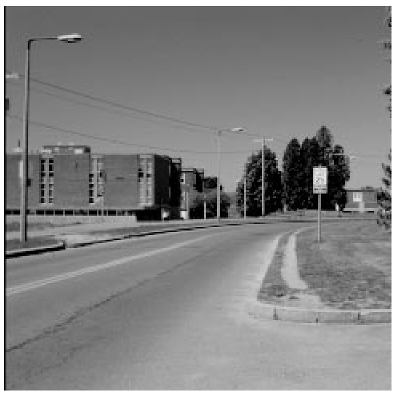

Figure 4: The first of twenty-one images taken from the same approximate viewpoint showing the Marcus Engineering building.

### 3.4.1 Knowledge Engineering

From the knowledge engineering perspective, preparing the strategy required fewer than three days of effort by a single person. One day of this was spent establishing the ground truth data that would be used to judge which hypotheses were correct and which were not. Another day was spent constructing a wire-frame model of the building's shape to be used as a compile-time argument to the pose determination ([9]) and geometric model

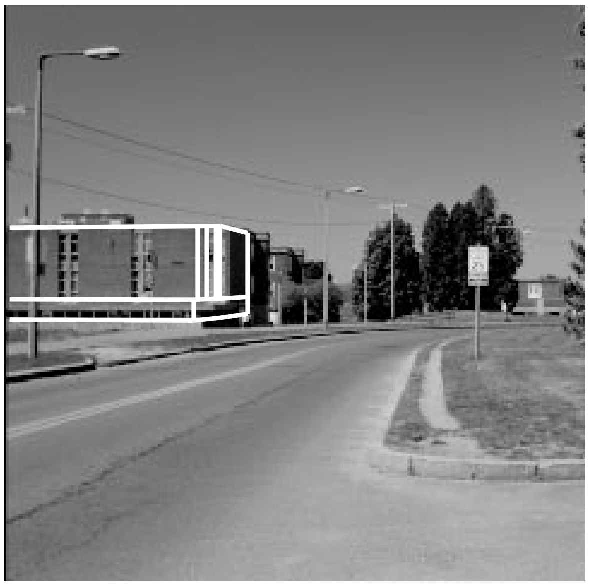
Figure 5: The same image as Figure 4, with the building pose found by SLS overlaid on it.

matching ([1]) knowledge sources. Finally, part of a third day was spent parameterizing and/or slightly modifying general-purpose knowledge sources.

Overall, the experiment can be considered a success from the knowledge engineering perspective. Not only was the knowledge base development time reduced to under three days, but most of that time was spent on tasks that can be easily automated. Ground truth, in particular, can be established to within acceptable limits by laser range finders or other direct 3D-imaging devices when the training images are acquired, and in many industrial applications shape models can be transfered directly from existing CAD/CAM databases. It therefore seems reasonable that object recognition strategies can be acquired in less than a day.

### 3.4.2 Experimental Robustness and Efficiency

The aim of the experiment was to test both the robustness of the strategies learned by SLS and the computational complexity of SLS itself. The later point, in particular, was of concern: in the worst case, the algorithms for reducing dependency trees to DNF format can produce exponentially more terms than there are training samples, but the worst case corresponds to essentially random data. As long as the training sample are self-similar this case should never arise; in fact, we expected that the observed complexity measured as the number of terms in the DNF expression should be nearly linear in the number of training samples, since each additional training sample is less likely than its predecessor to alter the final DNF expression.

Robustness was tested by a "leave N out" scheme of testing. We began by training a strategy on thirteen of the images and testing on the fourteenth, and repeating this process fourteen times. Not only did SLS learn a strategy which succesfully generated a correct hypothesis for the test image every time, but it learned the same strategy in all fourteen cases. (The strategy's ability to verify hypotheses was not tested.)

The experiment was then repeated using smaller and smaller sets of images as training data, training initially on twelve images, then on eleven, and so on all the way down to the ridiculous extreme of training a strategy on a single image and testing it on the other thirteen. Figure 6 shows the results of running fourteen tests for each number of training samples; in this domain, eight training images were enough for SLS to learn a strategy that reliably generated correct hypotheses, and even strategies infered from only seven training images could generate a correct building pose hypothesis 95% of the time.

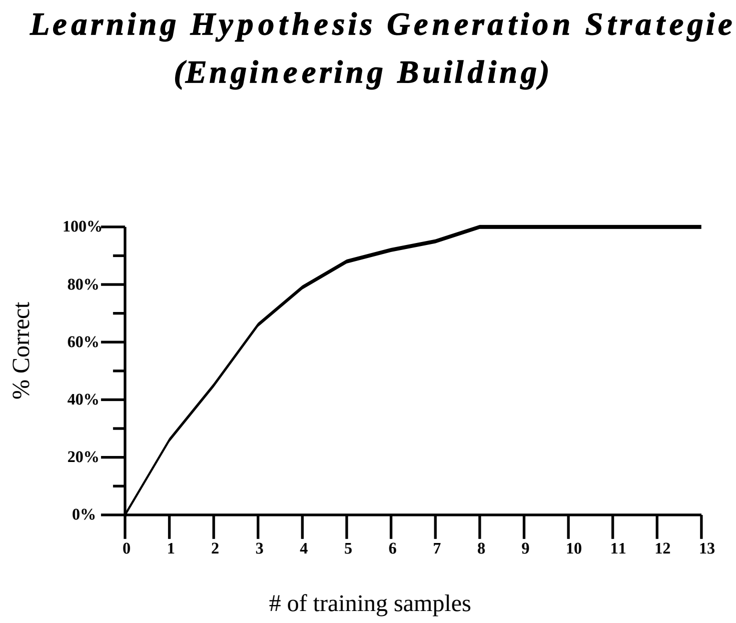

Figure 6: Performace of the strategy learned by SLS for recognizing a particular building from a known viewpoint. The horizontal axis shows the number of images in the training set, while the vertical axis shows the percentage of test images in which the correct hypothesis was generated.

Throughout this experiment, the complexity of the algorithm matched our predictions. The number of terms in the DNF expression, instead of growing exponentially with the number of training samples as the worst-case analysis predicts, was an approximately linear function of the number of training samples; in fact, the slope of the line was negative, and in no case did the final DNF expression have more than six terms. Figure 7 shows the average number of terms in the DNF expression for each training set size.

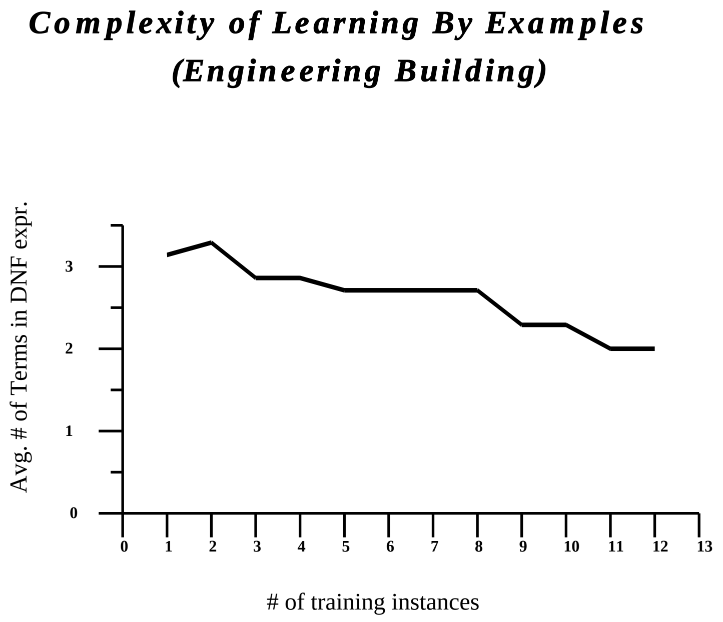

Figure 7: The observed complexity of learning by examples (average). The horizontal axis shows the number of images in the training set, while the vertical axis shows the average number of terms in the resulting DNF expression.

## 4 Conclusion

It is generally accepted that the timely and appropriate use of relevant knowledge can substantially reduce the search encountered in establishing 'instance-of' relationships between image data and its interpretation(s). This premise is supported by our experience with the schema system, a system that used object-specific knowledge to interpret road and house scenes.

Unfortunately, the problem of how to acquire and structure knowledge has limited most knowledge-based vision systems to highly constrained domains. The schema learning system (SLS) is an experimental system that learns how to recognize objects from training images. The eventual goal is to completely eliminate the knowledge engineering task.

At the moment, SLS still requires a human to supply it with parameterized knowledge sources and a set of interpreted training images. The previously time-consuming process of supplying control knowledge, however, has been eliminated, reducing the time specify and test a new object description to about one day. SLS learns its own control strategies that minimize the expected cost of recognition. Moreover, since visual control strategies are often counterintuitive, the interpretation strategies learned by SLS often outperform interpretation strategies designed by hand.

## References

[1] J. Ross Beveridge, Richard Weiss and Edward M. Riseman. "Optimization of 2-

Dimensional Model Matching," *Proc. of the DARPA Image Understanding Workshop,* Palo Alto, CA., June 1989. Morgan-Kaufman Publishers, Los Altos, CA. pp. 815-830.

[2] John Brolio, Bruce A. Draper, J. Ross Beveridge and Allen R. Hanson. "The ISR: an intermediate-level database for computer vision", *Computer*, 22(12):22-30 (Dec. 1989).

[3] Bruce A. Draper, Robert T. Collins, John Brolio, Allen R. Hanson and Edward M. Riseman. "The Schema System," *International Journal of Computer Vision,* 2:209–250 (1989).

[4] Bruce A. Draper and Edward M. Riseman. "Learning 3D Object Recognition Strategies", *3rd International Conference on Computer Vision*, Osaka, Japan, Dec., 1990. pp. 320-324.

[5] B.A. Draper. *Learning Object Recognition Strategies,* forthcoming Ph.D. dissertation, Univ. of Massachusetts.

[6] Allen R. Hanson and Edward M. Riseman. "VISIONS: A Computer System for Interpreting Scenes," in *Computer Vision Systems,* Hanson and Riseman (eds.), Academic Press, N.Y., 1978. Pp. 303-333.

[7] Allen R. Hanson and Edward M. Riseman, "The VISIONS image understanding system – 1986" in *Advances in Computer Vision,* C. Brown (ed.), Erlbaum: Hillsdale, N.J., 1987.

[8] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research.* Holden-Day, Inc: San Francisco, 1980.

[9] Rakesh Kumar and Allen R. Hanson. "Robust Estimation of Camera Location and Orientation from Noisy Data with Outliers," ??.

[10] Takashi Matsuyama. "Expert Systems for Image Processing: Knowledge-Based Composition of Image Analysis Processes" *Computer Vision, Graphics, and Image Processing,* 48:22–49 (1989).

[11] David M. McKeown, Jr., Wilson A. Harvey, and Lambert E. Wixson. "Automating Knowledge Acquisition for Aerial Image Interpretation" *Computer Vision, Graphics, and Image Processing,* 47:37–81 (1989).

[12] H. Penny Nii. "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *AI Magazine,* 7(2):38-53 (1986).