A Framework for Multi-Domain Sketch Recognition

Christine Alvarado, Michael Oltmans and **Randall Davis**
MIT Artificial Intelligence Laboratory
{calvarad,moltmans,davis}@ai.mit.edu

Abstract

People use sketches to express and record their ideas in many domains, including mechanical engineering, software design, and information architecture. Unfortunately, most computer programs cannot interpret free-hand sketches; designers transfer their sketches into computer design tools through menu-based interfaces. The few existing sketch recognition systems either tightly constrain the user's drawing style or are fragile and difficult to construct. In previous work we found that domain knowledge can aid recognition. Here we present an architecture to support the development of robust recognition systems across multiple domains. Our architecture maintains a separation between low-level shape information and high-level domain-specific context information, but uses the two sources of information together to improve recognition accuracy.

Introduction

Sketching is an efficient way to convey and record both physical information, such as mechanical engineering designs, and conceptual information, such as information architectures. Because sketching is simple and fast, it is often used in the early stages of design (Ullman, Wood, & Craig 1990).

Most computer aided design tools target the later, more stable stages of design instead of the earlier, more conceptual stages. Their domain-specific representation of the user's design gives them power, such as the ability to generate skeleton code from a UML diagram[1], but their interfaces constrain the user to picking from a menu of pieces. Therefore, designers often sketch their early designs on paper and transfer them to the computer later in the design process. Not only is this transfer a time-consuming process, but a design system's inability to interpret sketches prevents it from understanding the design at a time when the designer is making many important design decisions.

Imagine instead that the designer could sketch directly onto the computer using a digitizing tablet without changing her natural drawing style. While she drew, a tool would recognize her strokes as objects in a specified domain. At any time during the design process she would have the freedom of sketching on paper and the benefit of a computer aided design tool.

One of the most difficult problems in creating such a sketch recognition system is handling the tradeoff between ease of recognition and drawing freedom. The more we constrain the user's drawing style, the easier recognition becomes. For example, if we enforce the constraint that each component in the domain must be drawn with a carefully constructed symbol that can be drawn with a single stroke, we can build recognizers that can distinguish between the symbols, as was done with the Palm Pilot™. The advantage in using single-stroke recognizers is accuracy; the disadvantage is the designer is constrained to a specific language.

In response, researchers have suggested several approaches for building computer design tools for early stages of design. One approach is to focus away from recognition and build tools that support the design process without worrying about interpreting the sketch as suggested by Landay (2001). While this approach does involve the computer in the early design, making it easy to record the design process, in some domains it does not lend itself to the automatic transition from the early stage design tool to a more powerful design system.

A second approach is to carefully design a constrained set of icons or menus that, while constraining the user's drawing style, do not inhibit the design process. This approach should not be ruled out and eventually must be tested against free-sketch systems.

We focus on a third approach—a recognition system that can interpret the designer's sketches without disrupting the drawing process. The ideal recognition system would behave as a human observer would, looking over the designer's shoulder, unobtrusively interpreting her sketch as she draws. Such a recognition system should have the following characteristics:

- **Recognition accuracy and awareness**: If the user has to stop to correct the system's interpretations, it will interrupt the design process. Similarly, if the system tries to interpret the user's strokes before it has enough information, it is more likely to err in interpreting pieces of the user's drawing. The system should thus not only be very accurate with its recognition, but also be aware of when it has enough information to interpret a piece of the drawing

[1]UML stands for Universal Modeling Language, and is a visual language for designing object oriented software systems.

and when it should wait for more strokes to be drawn.

- **Easily implemented in most domains**: Recognition systems are currently tedious to construct and must be engineered from scratch for each particular domain. A good framework should be able to apply the same set of low-level geometric recognizers and generic recognition techniques to a variety of domains.

This paper presents a domain-flexible architecture for sketch recognition systems that we believe will make these systems both easier to construct and more robust in operation. At the heart of our approach is a method that combines high-level, domain-specific information with low-level, domain-independent recognizers.

The structure of this paper is as follows. First, we describe the difficulties involved in recognition and explore the power that comes from domain-specific information. Next, we describe the type of knowledge used in our framework and how that knowledge is used for recognition. In the next section we detail the structure we use to implement this recognition process. We then discuss the advantages of our approach, as well as the open questions and future directions. Finally, we discuss related work.

A Closer Look at Recognition

At the simplest level, recognition involves observing the outside world, and matching these observations to previously seen, named patterns. Thus, a recognition system consists of two separate pieces: a representation of the knowledge of the world around it, such as the shapes it can recognize, and a method for applying that knowledge to incoming data. The difficulty in building a recognition system lies in determining what knowledge to include in our representation and how to apply this knowledge.

A Simplistic Model

Traditional sketch recognition approaches, such as the approach developed by Rubine (1991), are powerful for single stroke classification; however, they do not scale well to complex sketches because they make the assumption that patterns in a domain can be represented and recognized in isolation.

Assuming that shapes can be recognized independently allows for construction of a single isolated model for each shape in a domain. Systems built on this assumption apply each shape model to new input to determine which shape most closely describes the input. These models are relatively straightforward to build and apply, and in simple domains they are fairly accurate.

Unfortunately, this approach has some serious consequences. In order to perform recognition, the system must determine which strokes should be grouped to form valid patterns. To avoid this problem, some systems constrain the user to drawing each object with a single stroke. While this constraint may be fine for editing gestures or simple shapes such as circles or lines, it quickly begins to limit the designer's freedom. Imagine trying to draw a pulley with a single stroke! Eventually the user is forced to learn a specialized alphabet just to be able to draw.

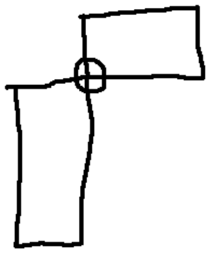


Figure 1: A mechanical engineering sketch. Context helps us interpret the circle as a pivot joint connecting two mechanical bodies.

While stroke grouping is difficult, there is a more serious problem with the isolated recognition approach—the system loses the ability to use the context in which that shape appears to help recognize the shape. Context is not merely helpful, it is essential to robust recognition of unconstrained sketches.

Context

Recognition is more than just identifying shapes in isolation. The context in which a particular stroke or group of strokes appears influences the interpretation of that stroke. Hence, recognition involves both the identification of patterns and the interpretation of those patterns with respect to the context in which they appear.

Context influences recognition in two distinct ways: disambiguating between possible interpretations and helping to recognize shapes that otherwise would have been misrecognized. In previous work we have shown that domain knowledge is essential to resolving inherent ambiguities in a sketch (Alvarado & Davis 2001). For example, consider the sketch in Figure 1. Imagine that the rectangular shapes can only be recognized as mechanical bodies. The circle, on the other hand, could either be a mechanical body or a pivot joint connecting the two rectangular bodies. By its shape we cannot determine the correct interpretation, but when we note that in the domain of mechanical engineering bodies are likely to be connected with joints, and unlikely (in our two-dimensional world) to overlap with other bodies without a connection, we can conclude that the circle is likely to be a pivot joint. Context also helps us identify messy input. Consider the shape connecting the two boxes in Figure 2. In isolation, it might be difficult to identify, but if we realize that the boxes are classes in a UML diagram and we know that classes are linked together with arrows, it immediately becomes clear that the correct interpretation for that shape is an arrow.

While context is essential to robust recognition, it is necessary to maintain a separation between shape recognition and contextual influence in order to build extendable systems. A naive approach is to build context directly into the pattern models in a given domain. For example, a UML recognition system might represent an arrow as a stroke that connects two classes and that is more or less straight. This

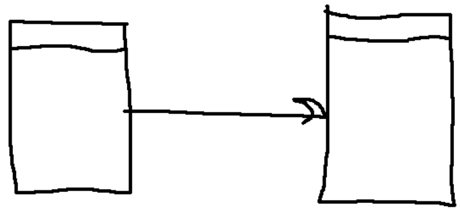


Figure 2: A UML diagram. Context helps us recognize the stroke between the two boxes as an arrow, even though it might not look like one in isolation.

model works for the domain of UML, but cannot be applied to other domains (for example, in mechanical engineering where arrows represent forces). The arrow itself is simply a shape whose canonical representation does not change from domain to domain; it is our tolerance for variations in its shape that varies according to the domain. Thus, basic shape models should be applicable across domains. The rest of this paper describes how our approach combines shape and contextual information to perform recognition in a given domain.

Hierarchical Recognition

Another source of power in human recognition is the ability to recognize objects hierarchically. Biederman (1987) suggests that humans recognize a set of primitive shapes, called geons, that combine to allow us to recognize more complicated objects.

Hierarchical recognition is not strictly necessary. For example, a system could represent an "X" as two crossed lines that bisect one another, or it could treat the image as a vector and learn a direct representation for an "X." However, hierarchical recognition provides a natural way to build an extendable recognition system. A system that recognizes an "X" directly from an image vector does not exploit the fact that it may already know how to recognize lines. On the other hand, a hierarchical system can use this fact simplify the recognition model for an "X," making it easier to specify. Furthermore, many sketched symbols are compositional; hence hierarchical recognition fits the task of sketch recognition.

Recognition Within Our Framework

We aim to build a recognition framework that can be applied naturally and efficiently to a variety of domains, yet takes advantage of the power that comes from context. We are motivated by the utility of domain knowledge for speech understanding, and inspired by the design of the Hearsay-II System (Erman *et al.* 1980). Hearsay-II combined knowledge at various levels of the speech interpretation process, including the syllable, word, and phrase levels, to generate and choose from multiple interpretations of a spoken utterance. We believe a similar architecture can be effective in sketch understanding.

As described above, recognition is the process of applying some pre-existing knowledge to identify new input. Recognition involves not only recognizing an object's shape, but also interpreting that shape within the surrounding context. Here we describe in more detail the types of knowledge our system uses to do recognition and describe our method for applying that knowledge to generate correct interpretations, despite messiness in the sketch and ambiguities in the drawing.

Knowledge Representation

To be easily extendable to different domains, our framework must be able to represent both general and domain-specific patterns. Given the demonstrated importance of domain-specific context for robust recognition, it must also be able to represent contextual information.

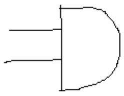
Because we are using a hierarchical recognition approach, we represent shapes using a shape description language. Shape description grammars were introduced formally by Stiny and Gips (Stiny & Gips 1972) and have been used mainly for generation of patterns. While they fell out of favor for pattern generation, we believe they are a feasible approach for recognition. As the focus in this paper is on recognition, not the shape description language, our treatment of the language is from the standpoint of its use as a knowledge representation. We do not formally analyze its theoretical properties.

At the lowest level, we define *primitive objects* to be patterns that cannot be recognized as a combination of other objects and must be recognized directly, such as lines and curves.

Objects within our recognition system have certain *properties* that describe possible interactions between them. We distinguish between three types of properties. A *condition* is a boolean property of an object or pair of objects, such as "vertical line1" or "concentric circle1 circle2." An *attribute* is a numerical measure of some aspect of an object or pair of objects, such as "length line1" or "angle-between line1 line2." A *comparison* is a relationship (e.g. greater-than, less-than or equals) between two attributes.

Finally, *compound objects* are constructed from primitive objects or other compound objects. They are specified by their component objects and properties of those component objects. For example, the specification for an and-gate is shown in Figure 3.

We draw a distinction between geometric objects, such as lines, circles, and arrows, and domain-specific objects such pivot joints and springs. This distinction is a subtle but important one. Whenever possible, we separate geometric patterns from their domain-specific counterparts. For example, in mechanical engineering, a force is represented with an arrow. However, because the arrow is also used in other domains, its shape is not specific to the domain of mechanical engineering. We would like to be able to specify information about a "force" without having that information tied to the geometric object "arrow". Therefore, we can specify a geometric compound object called an arrow that is made up of three lines and some properties of those lines, and a domain-specific compound object called a "force" that is just made up of an arrow. Thus we can use the arrow's shape model in other domains.



DEFINE AND-GATE

line L1 L2 L3	connected A.p2 L3.p2
arc A	meets L1.p2 L3
parallel L1 L2	meets L2.p2 L3
same-horiz-position L1 L2	semi-circle A
same-length L1 L2	orientation(A, 180)
connected A.p1 L3.p1	vertical L3

Figure 3: The description of an and-gate symbol includes the properties and low-level shapes that compose it.

In addition to representing shape information, we also represent contextual information. Our system uses two types of context to aid recognition. *Temporal context* provides information about the order in which components are likely to be drawn. For example, in mechanical engineering, bodies are usually drawn before the joints that connect them. *Spatial context* provides information about the shape patterns that are likely to occur in a given domain. For example, in UML diagrams, "instance-of" relations (represented with arrows) often link classes. While shape information can be domain-independent, contextual information is almost always specific to a particular domain.

Requirements for Recognition

In the next stage of the recognition process the system applies its knowledge of shape and context to produce the best mapping from domain-specific patterns to the user's strokes. Three challenges that must be addressed during this stage are segmentation, ambiguity resolution, and uncertainty management.

The process of segmentation involves determining which parts of a visual scene belong to which conceptual object. Because we wish to allow the user to draw objects freely, using any number of strokes in any order, our system must address this issue. Fortunately, our task is simplified because the system knows *a priori* the objects it is trying to identify, and each stroke usually part of only one object. Still, the process of finding the correct segmentation can be time consuming. Humans identify properties such as "tangent" or "parallel" without effort, but computer programs might have to consider many pairs of objects before discovering two with a given property. In general, this process will not work if applied naively: The combinatorics indicate that the system cannot simply try grouping each stroke with all other strokes before deciding on the best segmentation.

Another recognition difficulty is that ambiguities arise during the sketching process because shape information is not sufficient to identify a particular object. As we have seen, in Figure 1 it is not clear whether the circle is a mechanical body or a pivot joint.

Finally, the algorithm must be capable of handling uncertainties that arise during recognition due to messy and ambiguous input and deciding on the best interpretation for the user's strokes. Our algorithm must not allow the number of possible interpretations to grow too large, but it must also be sure not to eliminate any interpretation that might be correct.

Recognition Algorithm

In order to resolve the above problems, as the user draws, our algorithm generates a number of possible interpretations by combining bottom-up pattern activation with top-down knowledge application. We define an *interpretation* as a mapping from a set of strokes to a single high-level pattern. These interpretations are then pruned using the notion of "islands of certainty" developed in Hearsay-II.

There are four steps in our recognition algorithm:

1. **Bottom-up Step**: As the user draws, the system parses the strokes into primitive objects using a domain-independent recognition toolkit developed in previous work (Sezgin 2001). Compound interpretations are hypothesized by instantiating a template for each compound object that include these low-level shapes, even if not all the subcomponents of the pattern have been recognized.

2. **Top-down Step**: Once the system has a number of partially filled templates, it then identifies the missing subcomponents from these templates and attempts to reinterpret strokes that are temporally and spatially proximal to the proposed shape to fulfill the role of the missing components. If, for example, the system had detected an arc and two wires of the and-gate in Figure 3, it would try to reinterpret spatially and temporally adjacent strokes as lines to complete the gate.

3. **Ranking Step**: At this point, many different interpretations may have been proposed for the same strokes, some more likely than others. Based on previously interpreted parts of the sketch, the system identifies temporal and spatial context for the newly recognized patterns and uses this context to assign likelihoods to the templates that were generated in step 1 and modified in step 2. The system then explores sets of interpretations for the user's strokes starting with the highest ranked individual interpretation (an island of certainty) using a best-first-search method until it generates a given number of possible sets.

4. **Pruning Step**: Once the system has evaluated the likelihood of the various interpretations, it must prune off the interpretations that are unlikely to occur. The system makes concrete any interpretations that have likelihood above a threshold and eliminates any interpretations not appearing in the sets generated in step three. All other interpretations are deemed possible and are considered in relation to the user's next strokes when step 1 repeats.

In our initial implementation these steps will be applied in order; however, eventually our framework will use an opportunistic method of recognition in which these steps may be applied in any order, depending on the state of the recognition process.

Implementation

We use a blackboard architecture similar to the one used in Hearsay-II's speech recognition engine. A blackboard style implementation allows our system to combine multiple sources and types of knowledge into a unified framework. Unlike the Hearsay-II system, our system uses a Bayesian network to manage the uncertainties the arise during the recognition process.

A Blackboard Architecture

The idea behind a blackboard style architecture is simple. Imagine that to solve a problem you are going to bring in a bunch of specialists who all know something about a small piece of the problem. Together they can solve this problem, but they are not allowed to talk to one another. What do they do? Luckily, the problem is written on a blackboard in the front of the room, and the experts are allowed to go to the board one at a time, make some progress on the solution and then sit back down, leaving their improvement for all the other experts to see.

A blackboard system typically contains three major components: the blackboard data structure, the knowledge sources, and the controller. The blackboard holds the data relevant to the emerging solution. The knowledge sources are the specialists that each know a specific way of manipulating certain types of data on the blackboard. The controller is in charge of organizing the knowledge sources so that they make efficient progress toward the solution. A simplistic controller just lines up the knowledge sources and lets them each have a chance, while a more sophisticated controller selects knowledge sources that will contribute the most information to the problem solution.

Blackboards have several qualities that make them applicable to our framework. First, they allow us to easily extend and change the knowledge the system uses to do recognition. We can model both general and domain specific shape and context information inside knowledge sources. Because the knowledge sources do not depend on one another directly, if we move to a different domain, we can simply deactivate some knowledge sources and activate others. Second, blackboard architectures support system prototyping. As we implement our framework, we will be able to easily modify pieces of the recognition process and judge how each modification affects the system's performance. Finally, the blackboard data structure can be organized into different levels of information. The multi-layered structure supports our bi-directional recognition algorithm.

To illustrate information flow through the levels of the blackboard, consider what happens when a simplified version of our recognition algorithm is applied to the sketch in Figure 4. In the bottom-up step, six of the seven lines are recognized correctly. The three on the left are combined to form an arrow, which in turn is interpreted as a force. The four strokes on the right are not recognized directly as a polygon, but because the system sees that the three lines that were recognized have some of the properties of a polygon, it hypothesizes that a polygon exists. In the top-down step the system sees that it has a potential polygon, and looks for a stroke that might be the missing line. It finds the appropriate stroke and fits it into the polygon template. In step three, the fact that force arrows are often connected to bodies (spatial context) further enforces our belief that the stroke is indeed a line that is part of a polygon that represents a mechanical body.

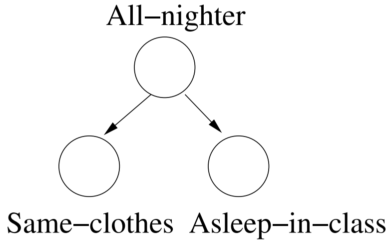


Figure 5: A simple Bayesian network. The fact that a student pulled an all-nighter makes it more likely that he will be wearing the same clothes the next day and that he will fall asleep in class.

Representing Uncertainty

We have indicated that our system must maintain a number of different interpretations for the user's sketch and that its ultimate goal is to choose the "most likely" set of interpretations. Our system uses a Bayesian network to represent recognition uncertainties.

Bayesian networks represent the influence one event has on another. They are structured as directed acyclic graphs. An arrow between two nodes indicates that that there is a causal relationship between the first event and the second. Each link contains a conditional probability table specifying how likely it is that one event caused the other. For example, we could model the consequences that pulling an all-nighter has on a student's ability to stay awake in class, and his tendency to wear the same clothes two days in a row (Figure 5). Ordinarily, the two events would not necessarily be related, but if we believe that pulling an all-nighter is a good predictor for both events, and we observe the student wearing the same clothes two days in a row, we might expect that he pulled an all-nighter the night before and therefore is likely to fall asleep in our class.

Each compound object is modeled by a fragment of a Bayesian network. The root node represents the compound object, and it has one child to represent each subcomponent and property. The representation for an and-gate is shown in Figure 6. We can think of this as similar to the all-nighter example. An and-gate is made up of certain components and relationships between those components. Thus, an and-gate can be considered a strong predictor of its subcomponents; if we see an and-gate, it is almost certain that we also see the arc and lines that make up that and-gate. However, not all the subcomponents of the and-gate are strong predictors for the and-gate; some carry more weight than others. This has to do with the probability that these components occur in other patterns in the sketch. For example, if we just see a line, our belief that the user is drawing an and-gate goes up only slightly, because lines are so common. However, if we

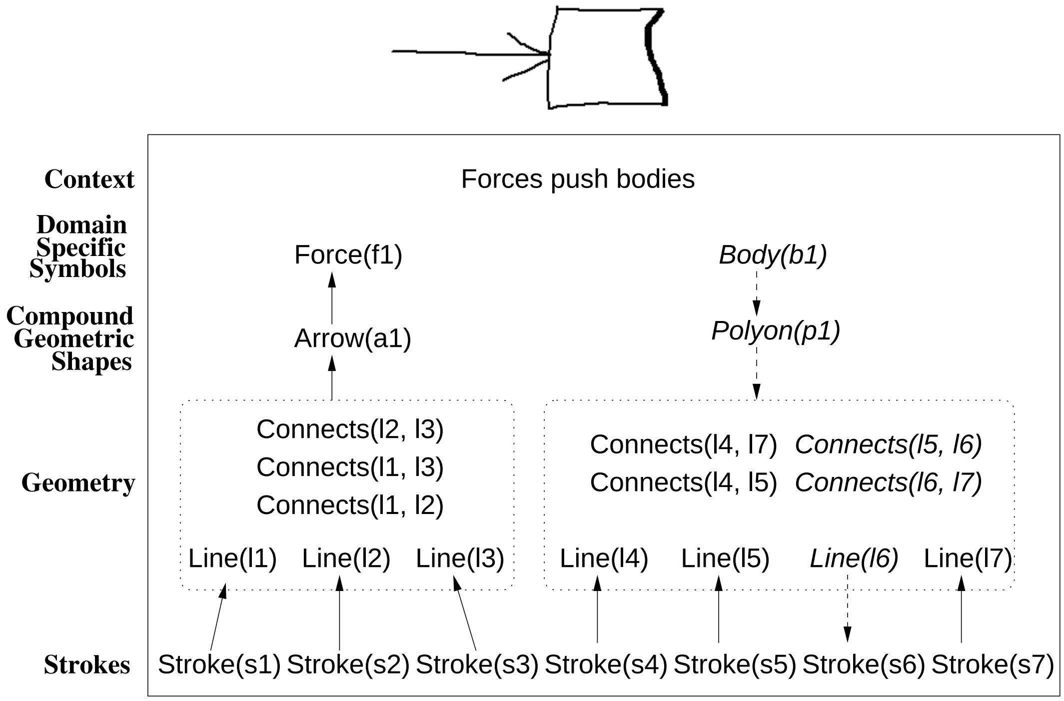


Figure 4: The recognition process in the drawing of a force pushing on a rectangular body. The blackboard data structure is divided into various levels of information. Italics indicate that an object was hypothesized by the system. In most cases, bottom up information causes the recognition of higher level objects, but in the case of the mechanical body, top down information reinforces the system's hypotheses.

see two parallel lines of the same length that meet a perpendicular line, we begin to be more sure of our prediction.

Representing the structure of complex objects using the nodes of a Bayesian Network seems straightforward at first, but not all aspects of this representation are so simple. To illustrate the problems we encounter and our solutions for each of these issues, we use the task of recognizing the play button for a CD player (Figure 7). The button is made up of a rectangle and a triangle and some properties of those shapes (i.e., the triangle is inside the rectangle and centered within it). However, the triangle and the rectangle are themselves compound shapes made up of lines and properties of those lines.

Ordinarily Bayesian networks represent a fixed set of information about the world. In a sketch, however, the information is constantly changing as the user adds and removes strokes. To address the issue of using Bayesian Networks in a dynamic environment, our system builds networks on the fly. For example, once the user has drawn the first three strokes of the rectangle, the system might decide to hypothesize the existence of a rectangle and instantiate the rectangle's Bayesian network fragment. This, in turn, leads the system to hypothesize the existence of a play button. The play button's Bayesian network fragment is instantiated using the root node of the rectangles fragment as one of its sub-components (see Figure 7).

Note that when the system hypothesizes fragments of the network, some of the leaf nodes, which always represent primitive objects or properties, may not be grounded in real data. For example, the system may not have seen the fourth line in the rectangle, or may not yet have calculated whether the triangle is inside the square. This is perfectly acceptable in Bayesian networks; those nodes are simply treated as "unobserved." However, when the system detects another primitive object or property to fit into the network, it must be able to attach a certainty measure to its observation. For example, the system might think it sees a line, but not be sure, as the stroke could also be an arc. Unfortunately, Bayesian networks require that observations be either true or false. To get around this problem, when the system observes primitive object or properties, it extends node for observed the object or property with a child node whose value is observed to be true (see Figure 7). The system then constructs a conditional probability table for the new link that reflects the appropriate certainty in the object or property's node.

Finally, while it is relatively straightforward to specify the structure of these networks, determining the appropriate numbers to fit into the network can be more problematic. What exactly does each of the numbers mean? Let's examine the play button ($\mathtt{P}$) template once again. For each child node $\mathtt{c}$, we must specify $P(\mathtt{c}|\mathtt{P})$ and $P(\mathtt{c}|\neg\mathtt{P})$. The first probability refers to the probability that a rectangle is present when a play button is present. This probability should be quite high since we define a play button to be constructed using a rectangle. The meaning of the second probability is less clear. It refers to the probability that we are observing a

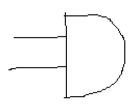
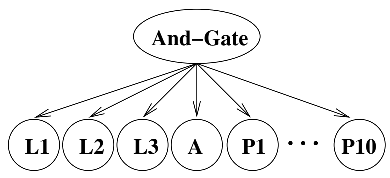
Sketch	Description	Network Fragment
	DEFINE AND-GATE *L1,L2,L3*: line L1 L2 L3 *A*: arc A *P1*: parallel L1 L2 *P2*: same-horiz-position L1 L2 *P3*: same-length L1 L2 *P4*: connected A.p1 L3.p1 *P6*: connected A.p2 L3.p2 *P5*: meets L1.p2 L3 *P7*: meets L2.p2 L3 *P8*: semi-circle A *P9*: orientation(A, 180) *P10*: vertical L3	

Figure 6: The description of an and-gate symbol from Figure 3. Each of these shapes and properties becomes a node in a Bayesian network fragment.

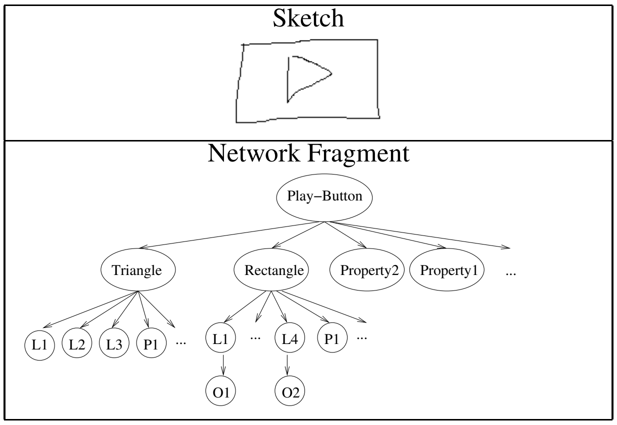


Figure 7: The sketch of a play button and its associated Bayesian network fragment. Note that this network fragment builds on top of a fragment for a rectangle and a fragment for a triangle. The leaf nodes (i.e. primitive objects and properties) that have been observed in the sketch are linked to observation nodes.

rectangle given that we know that the user is *not* drawing a play button. Intuitively this number is how likely a rectangle is to appear in other compound shapes in the domain scaled by how likely those shapes are to appear. Finally, we must specify the prior probability of seeing a play button. In our system, this number is not constant, but can be influenced by what else is going on in the sketch near the play button. For example, CD players rarely have more than one play button, so if the system has already recognized a play button, the prior probability of recognizing another one will go way down.

Discussion

We have not yet completed implementation of this framework, and therefore have not formally evaluated its strengths and weaknesses in practice. In this section we argue for the decisions we have made in designing this system and analyze the areas that require the most attention in future work.

Our framework has several recognition advantages over existing approaches. It uses contextual information to resolve ambiguities and handle messy input. To effectively apply context, it seamlessly integrates domain-independent and domain-specific information through a combination of bottom-up and top-down recognition. Bayesian networks are a natural tool for allowing low-level information to influence expectation of high-level components and in turn other low-level patterns. Furthermore, this framework is unique in its ability to apply domain-specific information from any domain and reuse domain-independent information.

Our approach is especially tailored to interactive recognition environments because at any point the system is aware of possible future interpretations of an incomplete sketch. For example, if the user has drawn a play button, the system knows to wait for a few more strokes before making this interpretation concrete because she might actually be drawing a fast forward button.

One important focus of our future efforts will concern resolving questions that arise within the Bayesian network representation. As one example, our system combines three types of knowledge by allowing the spatial and temporal context to alter the prior probabilities of the root nodes in the Bayesian network. Exactly how context should influence these probabilities is a non-trivial question at the heart of our approach that we will explore through experimentation.

Another problem arises in determining when exactly the system should instantiate recognition hypotheses. Just about every compound object contains a line; it seems wasteful to hypothesize every object each time the user draws a line. Determining how much information should be required to trigger each hypothesis is an area we must explore.

A final challenge is how to enter the knowledge into the recognition system. Grammars are tedious to write, and in previous work we found that explicitly specifying context, while possible, is difficult. Our group is currently investigating ways to learn the grammars (as Do and Gross (1996) have attempted), as well as the temporal and spatial information, from examples.

Related Work

Other sketch recognition systems include those developed by Landay and Meyers (2001), Do and Gross (1996), Forbus, Fergeson and Usher (2001) and Stahovich (1999). Each system copes with recognition ambiguity in a different way. Our previous work (Alvarado & Davis 2001) uses context to disambiguate between multiple interpretations of a sketch, but like other previous work it is still driven by low-level recognition accuracy. The work described here differs from all previous systems in its ability to allow high-level interpretations to guide low-level recognition accuracy.

Bimber *et. al.* (2000) have proposed a multi-layer architecture for recognizing sketches of three-dimensional systems. However, rather than focusing on more robust recognition to give the designer more drawing freedom, the authors focus on allowing users to specify two-dimensional gestures to represent three-dimensional solid objects.

Blackboard recognition systems were first introduced for the domain of speech recognition with the Hearsay-II system (Erman *et al.* 1980), and have since been extended to many other domains[2]. Unlike previous systems, we take a more structured approach to modeling the uncertainty throughout the recognition process with the use of Bayesian networks.

There has been a limited amount of work in using top down information to guide real-world visual interpretation including (Bienenstock, Geman, & Potter 1997), (Kumar & Desai 1996), and (Liang, Jensen, & Christensen 1996). Our domain is slightly less complex in that sketches are highly stylized, so the problem of locating (but not recognizing) low-level shapes is lessened.

Conclusion

We have described a framework that allows domain-specific shape and contextual information to be combined with generic shape information to make sketch recognition systems more robust and easier to construct. We claim that domain knowledge is essential to practical recognition systems.

Currently, recognition systems either constrain the user's drawing style or fail to robustly handle complex input. By incorporating domain knowledge in a structured fashion we believe we will be able to construct recognition systems with enough accuracy to be beneficial in early stages of design, allowing the designer to draw freely using computer design tools without modifying her drawing style.

Acknowledgements

This work is financially supported by the MIT-Oxygen Collaboration. The authors would like to thank Alex Snoeren for helpful editing comments.

References

Alvarado, C., and Davis, R. 2001. Resolving ambiguities to create a natural sketch based interface. In *Proceedings. of IJCAI-2001*.

Biederman, I. 1987. Recognition-by-components: A theory of human image understanding. *Psychological Review* 94(2):115–147.

Bienenstock, E.; Geman, S.; and Potter, D. 1997. Compositionality, mdl priors, and object recognition. In M. C. Mozer, M. I. Jordan, T. P., ed., *Advances in Neural Information Processing Systems 9*. MIT Press. 838–844.

Bimber, O.; Encarnacao, L. M.; and Stork, A. 2000. A multi-layered architecture for sketch-based interaction within virtual environments. *Computers & Graphics* 24:851–867.

Do, E. Y.-L., and Gross, M. D. 1996. Drawing as a means to design reasoning. *AI and Design*.

Erman, L.; Hayes-Roth, F.; Lesser, V.; and Reddy, R. 1980. The hearsay-ii speech-understanding system: Integrating knowedge to resolve uncertainty. *Computing Surveys* 12(2):213–253.

Forbus, K.; Ferguson, R.; and Usher, J. 2001. Towards a computational model of sketching. In *IUI '01*.

Kumar, V. P., and Desai, U. B. 1996. Image interpretation using bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18(1):74–77.

Landay, J. A., and Myers, B. A. 2001. Sketching interfaces: Toward more human interface design. *IEEE Computer* 34(3):56–64.

Landay, J. 2001. Informal tools for designing anywhere, anytime, anydevice user interfaces. Lecture at Stanford University, Seminar on People, Computers, and Design.

Liang, J.; Jensen, F. V.; and Christensen, H. I. 1996. A framework for generic object recognition with bayesian networks. In *Proceedings of the First Interational Symposium on Soft Computing for Pattern Recognition*.

Nii, H. P. 1986a. Blackboard application systems and a knowledge engineering perspective. *The AI Magazine* 82–107.

Nii, H. P. 1986b. The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine* 38–53.

Rubine, D. 1991. Specifying gestures by example. *Computer Graphics* 329–337.

Sezgin, T. M. 2001. Early processing in sketch recognition. Master's thesis, Massachusetts Institute of Technology.

Stahovich, T. F. 1999. Learnit: A system that can learn and reuse design strategies. In *1999 ASME Design Engineering Techincal Conference Proceedings*.

Stiny, G., and Gips, J. 1972. Shape grammars and the generative specification of painting and sculpture. In Freiman, C. V., ed., *Information Processing 71*. North-Holland. 1460–1465.

Ullman, D. G.; Wood, S.; and Craig, D. 1990. The importance of drawing in mechanical design process. *Computer & Graphics* 14(2):263–274.

[2] For for a comprehensive survey of blackboard systems see (Nii 1986a; 1986b).