Multi-Domain Sketch Understanding

by

Christine J. Alvarado

S.M. Computer Science and Engineering, Massachusetts Institute of Technology (2000)
A.B. Computer Science, Dartmouth College (1998)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2004

Author . . . .
Department of Electrical Engineering and Computer Science
August 19, 2004

Certified by . . . .
Randall Davis
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Multi-Domain Sketch Understanding

by

Christine J. Alvarado

Submitted to the Department of Electrical Engineering and Computer Science
on August 19, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

People use sketches to express and record their ideas in many domains, including mechanical engineering, software design, and information architecture. In recent years there has been an increasing interest in sketch-based user interfaces, but the problem of robust free-sketch recognition remains largely unsolved. Current computer sketch recognition systems are difficult to construct, and either are fragile or accomplish robustness by severely limiting the designer's drawing freedom.

This work explores the challenges of multi-domain sketch recognition. We present a general framework and implemented system, called *SketchREAD*, for diagrammatic sketch recognition. Our system can be applied to a variety of domains by providing structural descriptions of the shapes in the domain. Robustness to the ambiguity and uncertainty inherent in complex, freely-drawn sketches is achieved through the use of context. Our approach uses context to guide the search for possible interpretations and uses a novel form of dynamically constructed Bayesian networks to evaluate these interpretations. This process allows the system to recover from low-level recognition errors (e.g., a line misclassified as an arc) that would otherwise result in domain level recognition errors. We evaluated SketchREAD on real sketches in two domains—family trees and circuit diagrams—and found that in both domains the use of context to reclassify low-level shapes significantly reduced recognition error over a baseline system that did not reinterpret low-level classifications. We discuss remaining challenges for multi-domain sketch recognition revealed by our evaluation. Finally, we explore the system's potential role in sketch-based user interfaces from a human computer interaction perspective.

Thesis Supervisor: Randall Davis
Title: Professor of Computer Science and Engineering

Acknowledgments

I did not go to graduate school to work by myself, and I do not believe any that any work of this magnitude can be accomplished by an individual alone. It would be wrong for me to claim sole credit for the work in this dissertation, because the truth is that I could not have done it, nor would I have wanted to do it, without the help and support of many people.

First, I would like to thank my advisor, Randy Davis, for giving this work direction and for helping me maintain its direction over the course of six years. Six years ago, he asked me to work on a project that involved hooking a sketch recognition system into a mechanical simulator. I argued that linking two pieces of software was not research, and he replied that perhaps I was right, but that it would make "a hell of a cool demo." Not only was it an incredibly cool demo, but it also opened a door to years of new and exciting research. I owe the direction of this work to Randy's ability to see fascinating problems years before anyone else does.

The efforts of many people went into the creation of the software system that underlies work presented here. The motivation for SketchREAD was provided by an early sketch understanding system created chiefly by Luke Weisman. Luke was a wonderful person to work with and has since become a wonderful friend. Similarly, Mike Oltmans began graduate school with me, developed much code along with me, and has always been there for me as a colleague and a friend. I have also been lucky to have received many ideas and insights from the rest of the (perhaps improperly named) Design Rationale Group over the last six years: Aaron Adler, Oskar Bruening, Jacob Eisenstein, Mark Foltz, Tracy Hammond, Rebecca Hitchcock, Metin Sezgin, and Olya Vesselova.

I came to MIT because I was amazed by the energy and intelligence of the people here, and I have never ceased to be inspired by them. I would like to thank the other two members of my committee, Trevor Darrell and Leslie Pack Kaelbling. While I undoubtedly did not solicit their advice often enough, their insights were always quite helpful. David Karger, Mark Ackerman and Jaime Teevan gave me an unforgettable opportunity to pursue my interests outside of sketch recognition for which I am grateful. I would like to acknowledge also Mike Ross and Sameer Ajmani who started here at MIT with me and have been my friends throughout this process.

In a field that is so male dominated, I feel truly honored to have known so many wonderful and brilliant women in all areas of my life. Jaime Teevan and Tracy Hammond have been the best friends I could have hoped for and have inspired me both personally and professionally. Many of my hockey teammates, including Mary Krasovec, Raquel Romano, Susan (Dacy) Luschas, Sarah

Dolman, Vivian Lin, Jeanie Cherng, and Heather Gunter showed me that you can have a life *and* get a PhD. And although Jill Fekete didn't play with the club team long, I'll always remember our rides to and from Tuesday night hockey. The women of the WTP gave me hope that the field of Computer Science will not always be dominated by men. Finally, the women of 10 Chandler St. #2—Jessica Marchetti, Stacey Keating, Bennett Arble, Amy Antman, Erica Hatch, Kamila Pomicienska, and Alex Carter—have provided a safe haven over the years, away from the stresses of MIT. I would not have survived without this daily escape.

I am lucky to have such a wonderful and supportive family, and I credit them with getting me to where I am today. My sister, Liz, always has the courage to follow her dreams and I always remember that when I am afraid to follow mine. My father never doubted for a second that I could do this, and couldn't understand why I would ever doubt myself. And although both my parents have PhDs, it has been my mother, who inspired us all by getting her PhD just a year and a half ago, who has really been able to understand and support me through these last six months.

Finally, I am indebted beyond words to Alex Snoeren. Over the years he has been a partner, a colleague, a source of support, at times a challenge, and always and above all my best friend. I simply could not have done this without him. I am truly lucky that I will always have him as part of my family of two.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work addresses the challenges of creating a computer system capable of of understanding the informal, messy, hand-drawn sketches typically created during the design process. Our thesis is that high-level context can be combined with low-level stroke information to provide robust, multi-domain sketch recognition. We validate this thesis through the design and implementation of a system called SketchREAD (Sketch Recognition Engine for Any Domain) that "understands" a user's sketch in that it parses a user's strokes as they are drawn and interprets them as objects in a domain of interest. While previous sketch recognition systems have placed constraints on the user's drawing style or performed only limited sketch recognition, SketchREAD allows the user to draw freely and attempts to recognize all the symbols in the user's sketch. We evaluated SketchREAD on real sketches in two domains and show that the use of context can significantly improve recognition performance. Our evaluation also makes concrete a number of remaining challenges for two-dimensional multi-domain sketch recognition. For simple domains, SketchREAD's robustness and lack of constraint on drawing style allows us to explore the usability challenges of incorporating free-sketch recognition into diagram creation tools. We used SketchREAD to incorporate free-sketch recognition into a diagram creation tool for PowerPoint and present guidelines for creating this type of sketch recognition-based user interface.

1.1 Building a Natural and Powerful Sketch Tool

Designers in many disciplines use sketching to express their ideas throughout the design process. Sketching provides a lightweight and natural way to put down ideas during the early stages of design. Sketches can be used to record both physical designs (e.g., mechanical and electrical engineering designs) and conceptual designs (e.g., organizational charts and software diagrams). Computers have emerged as powerful design tools with at least one notable exception—they offer little or no support for free-hand sketching. As a result, people usually sketch their designs on paper first, transferring them to the computer only when they are nearly complete. There are two related reasons for this behavior. First, many designers do not use pen-based computer technology, and the mouse is a clumsy input device for free sketching; it is more suited to menu-based interfaces. Second, most diagram creation and design programs are not targeted to the early stages of design. Because they are targeted at the later stages of design, they assume users know exactly what they are drawing and force users to input their designs through a series of menus—an input style that is far from the freedom that paper provides a designer in part because it forces them to be precise in their design.

While in recent years there has been an increasing interest in sketch-based user interfaces [48, 63, 69], the problem of robust free-sketch recognition remains largely unsolved. Because existing sketch recognition techniques are difficult to implement and are error-prone or severely limit the user's drawing style, many previous systems that support sketching perform only limited recognition. ScanScribe, for example, uses perceptual guidelines to support image and text editing but does not attempt to recognize the user's drawing [69]. Similarly, the sketch-based DENIM system supports the design of web pages but recognizes very little of the user's sketch [63]. Systems of this sort involve the computer in the early design, making it easy to record the design process, but they do not always facilitate automatic transition from the early stage design tool to a more powerful design system.

A tool with the power of existing design tools but that is useful in early design should allow designers to sketch as freely as they would with pencil and paper, while offering considerable advantage over paper. The power of computer design tools comes from their ability to work with the user's designs as domain-specific diagrams. One way we can provide the same power in a sketch-based tool is to build a program that can understand the user's design in domain-specific terms as she sketches.

One of the most difficult problems in creating a sketch recognition system is handling the trade-off 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 a carefully drawn symbol that can be created with a single stroke, it is relatively easy to build recognizers capable of distinguishing between the symbols, as was done with Palm Pilot Graffiti. The advantage of using restricted recognizers is accuracy; the disadvantage is the designer is constrained to a specific style of sketching.

Previous recognition-intensive systems have focused on tasks where drawing style assumptions can greatly reduce recognition complexity. Long *et al.* focus on designing special graphical symbols that will not be confused easily by the computer [54]. This approach improves recognition, but it limits the designer to a specific set of single-stroke symbols that may be natural only for certain tasks. The Quickset system for recognizing military course of action (COA) diagrams uses multi-modal information to improve recognition of sketched symbols [83], but assumes that each symbol will be drawn independently, and that the user will likely speak the name of the symbol when drawing it. These assumptions aid recognition, but may fail for design tasks in other domains. In electrical engineering, for example, designers draw several symbols without pausing and probably do not speak the name of the symbols they draw. Other previous systems have similar constraints on drawing style or do not provide the level of recognition robustness we seek here [4, 73, 32].

While the previous systems have proven useful for their respective tasks, we aim to create a general sketch recognition system that does not rely on the drawing style assumptions of any one domain. To be usable, a sketch recognition-based system must make few enough mistakes that sketching is less work than using a more traditional (i.e., menu-based) interface. To be broadly effective the system's architecture should be easily applied across a variety of domains, without having to re-engineer the system.

In addition, there are a number of usability challenges in incorporating free sketch recognition into a design tool. These challenges include, but are not limited to:

- When and how to display recognition feedback
- How to allow the user to edit her diagrams
- How to allow the user to specify the recognition domain
- How to handle and allow the user to correct recognition errors

1.2 A Motivating Example

Like handwriting and speech understanding, sketch understanding is easy for humans, but difficult for computers. We begin by exploring the inherent challenges of the task.

Figure 1-1 shows the beginning of a sketch of a family tree, with the strokes labelled in the order in which they were drawn. The symbols in this domain are given in Table 1.1. This sketch is

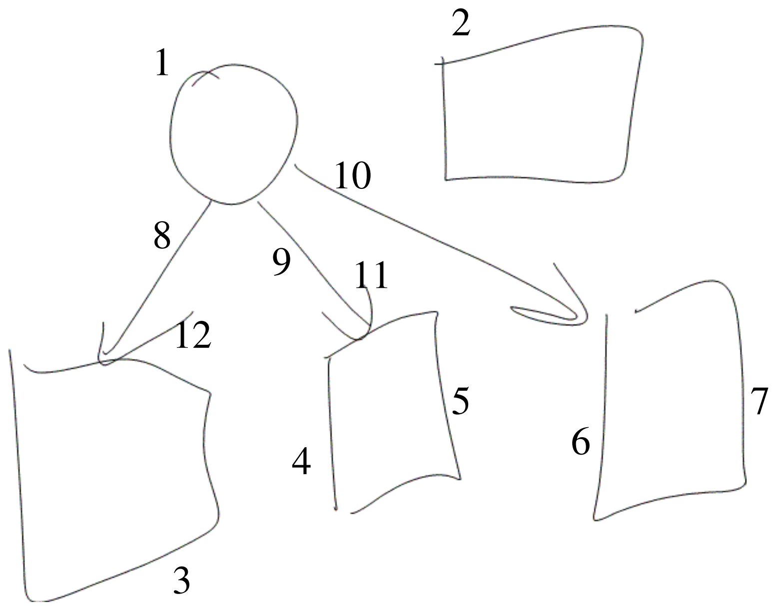
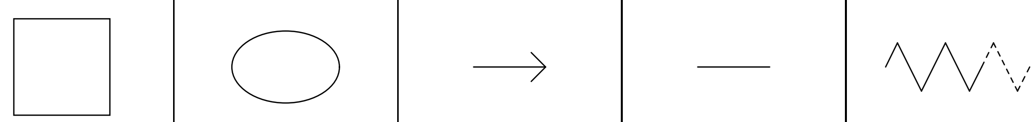


Figure 1-1: A partial sketch of a family tree.



Male	Female	Child-link	Marriage-link	Divorce-link

Table 1.1: The symbols in the family tree domain.

based on the real data collected from users shown in Figure 1-3, but has been has been redrawn to illustrate a number of challenges using a single example. The user started by drawing a mother and a father, then drew three sons. He linked the mother to the sons by first drawing the shafts of each arrow and then drawing the arrowheads. (In our family tree diagrams, each parent is linked to each child with an arrow.) He will likely continue the drawing by linking the father to the children with arrows and linking the two parents with a line.

Although relatively simple, this drawing presents many challenges for sketch recognition. While previous recognition systems address some of these challenges, our system is the first to address all of them using a general framework that can be extended to multiple domains.

The first challenge illustrated in Figure 1-1 is the incremental nature of the sketch process. Incremental sketch recognition allows the computer to seamlessly interpret a sketch as it is drawn and keeps the user from having to specify when the sketch is complete. To recognize a potentially incomplete sketch, a computer system must know when to recognize a piece of the sketch and when to wait for more information. For example, Stroke 1 can be recognized immediately as a female, but Stroke 6 cannot be recognized without Stroke 7.

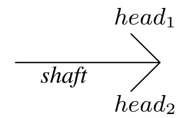
The second challenge is that many of the shapes in Figure 1-1 are visually messy. For example, the center arrowhead (Stroke 11) looks more like an arc than two lines. Next, the stroke used to draw the leftmost quadrilateral (Stroke 3) looks like it is composed of five lines—the top of the quadrilateral has a bend and could be reasonably divided into two lines by a stroke parser. Finally, the lines in the rightmost quadrilateral (Strokes 6 and 7) obviously do not touch in the top left corner.

The third issue is segmentation: It is difficult to know which strokes are part of which shapes. For example, if the computer knew that Strokes 9 and 11 were part of one shape, the system would likely be able to match an arrow pattern to these strokes using a standard algorithm, such as a neural network. Unfortunately, segmentation is not an easy task. The shapes in this drawing are not clearly spatially segmented, and naively trying different combinations of strokes is prohibitively time consuming. To simplify segmentation, many previous systems (e.g., [63, 83]) assume each shape will be drawn with temporally contiguous strokes. This assumption does not hold here.

There are also some inherent ambiguities in how to segment the strokes. For example, lines in our domain indicate marriage, but not every line is a marriage-link. The shaft of the leftmost arrow (Stroke 8) might also have been interpreted as a marriage-link between the female (Stroke 1) and the leftmost male (Stroke 3). In this case, the head of that arrow (Stroke 12) could have been interpreted as a part of the drawing that is not yet complete (e.g., the beginning of an arrow from the leftmost quadrilateral (Stroke 3) to the top quadrilateral (Stroke 2)).

Finally, how shapes are drawn can also present challenges to interpretation. The head of the rightmost arrow (part of Stroke 10) is actually made of three lines, two of which are meant to overlap to form one side of the arrowhead. In order to recognize the arrow, the system must know to collapse those two lines into one, even though they do not actually overlap. Another challenge arises because the same shape may not always be drawn in the same way. For example, the arrows on the left (Strokes 8 and 12, and Strokes 9 and 11) were drawn differently from the one on the right (Stroke 10) in that the user first drew the shaft with one stroke and then drew the head with another. This variation in drawing style presents a challenge for segmentation and recognition because a system cannot know how many strokes will be used to draw each object, nor the order in which the parts of a shape will appear.

Many of the difficulties described in the example above arise from the messy input and visual ambiguity in the sketch. It is the context surrounding the messy or ambiguous parts of the drawing



DEFINE ARROW
(**Subshapes**
(Line *shaft* $head_1$ $head_2$))
(**Constraints**
(coincident *shaft*.p_1 $head_1.p_1$) (larger *shaft* $head_1$)
(coincident *shaft*.p_1 $head_2.p_2$) (acuteAngle $head_1$ *shaft*)
(equalLength $head_1$ $head_2$) (acuteAngle *shaft* $head_2$))

Figure 1-2: The description of the shape "arrow." More details on this description are given in Chapter 2.

that allows humans to interpret these parts correctly. We found that context also can be used to help our system recover from low-level interpretation errors and correctly identify ambiguous pieces of the sketch. Context has been used to aid recognition in speech recognition systems; it has been the subject of recent research in computer vision [74, 75] and has been used to a limited extent in previous sketch understanding systems [4, 32, 63]. We formalize the notion of context suggested by previous sketch recognition systems. This formalization improves recognition of freely drawn sketches using a general engine that can be applied to a variety of domains.

1.3 Approach Overview

We use a probabilistic, hierarchical shape description language to describe the shapes in a domain. Each shape is composed of components and constraints between those components. Figure 1-2 shows a simple use of the language to describe an arrow (which can be used in a number of domains). This language is described in more detail in Chapter 2 and in previous work by Hammond and Davis [37]. A hierarchical representation is useful because it enables re-use of geometric shapes (e.g., rectangles and arrows) across a variety of domains and because many sketched symbols are compositional. The representation is probabilistic to reflect variation in the way objects are drawn, due both to noise in the data and individual drawing styles.

Recognizing the sketch is a matter of parsing a user's strokes according to the specified visual language. Visual language parsing has been studied [60], but most previous approaches assume diagrammatic input free from low-level recognition errors and cannot handle realistic, messy, stroke-based input. Mahoney and Fromherz use mathematical constraints to cope with the complexities of parsing sketches of curvilinear configurations such as stick figures [58]. Shilman *et al.* present a parsing method similar to our approach [72], with two differences. First, their work employs a spatially-bounded search for interpretations that quickly becomes prohibitively expensive. Second, their parsing method builds and scores a parse tree for each interpretation independently; we allow competing interpretations to influence each other.

As the user draws, our system uses a two-stage, generate-and-test recognition process to parse the strokes into possible interpretations. This two-dimensional parsing problem presents a challenge for a real-time system. Noise in the input makes it impossible for the system to recognize low-level shapes with certainty or to be sure whether or not constraints hold. Low-level misinterpretations cause higher-level interpretations to fail as well. On the other hand, trying all possible interpretations of the user's strokes guarantees that an interpretation will not be missed, but is infeasible due to the exponential number of possible interpretations.

To solve this problem we use a combined bottom-up and top-down recognition algorithm that generates the most likely interpretations first (bottom-up), then actively seeks out parts of those interpretations that are still missing (top-down). Our approach uses a novel application of dynamically constructed Bayesian networks to evaluate partial interpretation hypotheses and then expands the hypothesis space by exploring the most likely interpretations first. The system does not have to try all combinations of all interpretations, but can focus on those interpretations that contain at least a subset of easily-recognizable subshapes and can recover low-level subshapes that may have been mis-recognized.

1.4 Results Overview

We apply our recognition system to three domains: family tree diagrams, relationship diagrams, and electrical engineering diagrams. Tables 1.1, 1.2, and 1.3 list the shapes our system recognizes in each of these domains. Complete specifications for each of the three domains is given in Appendix A.

We collect and analyze real user data for both the family tree and electrical engineering domains. Figure 1-3 shows the range of sketches our system is able to recognize. For the family tree domain, SketchREAD correctly recognizes 77% of the symbols in the diagrams, which is 48% fewer recognition errors than a baseline version of SketchREAD that does not reinterpret low-level

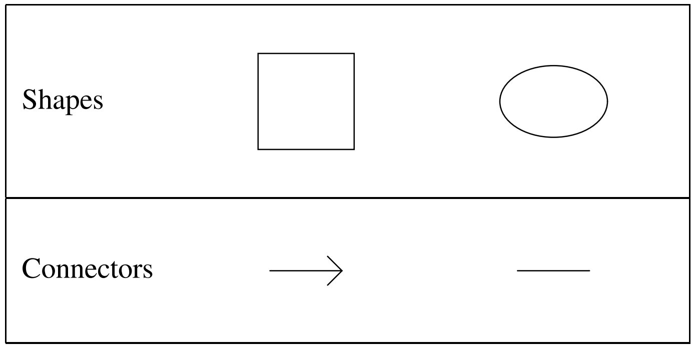


Table 1.2: The symbols in the Relationship Diagram domain. Boxes and ellipses represent entities that can be related using lines and arrows.

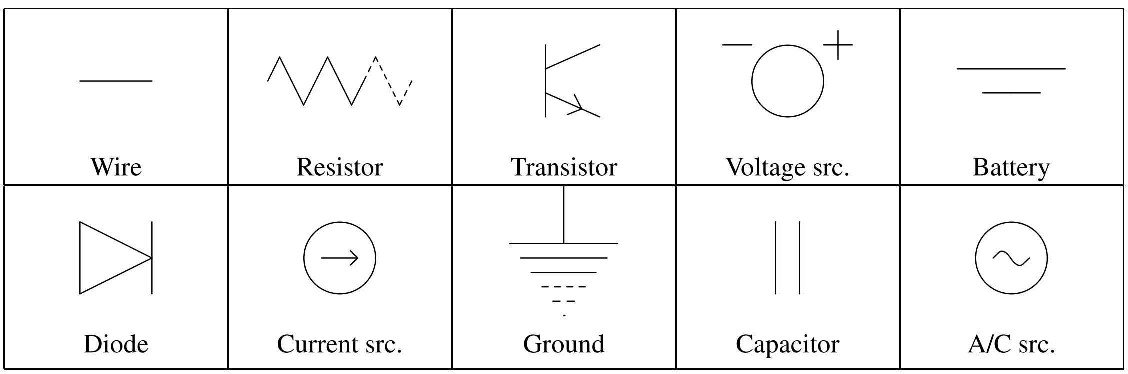


Table 1.3: The symbols in the circuit domain.

classifications. For the circuit domain, SketchREAD correctly identifies 62% of the total shapes, a 17% reduction in error over the baseline system. We also examine SketchREAD's running time to determine how it scales with the number of strokes in the sketch and find that the time to process each stroke increases only slightly as the sketch gets larger. However, we note that SketchREAD does not yet run in real time on complex sketches, and certain sketched configurations cause the system to run for a long time. We present more results, including both a qualitative and a quantitative analysis of the system's performance, in Chapter 5.

Using the relationship diagram domain, we implemented a sketch-based PowerPoint diagram-creation tool to explore the usability issues involved in creating sketch recognition user interfaces (SkRUIs). Based on iterative design and evaluation of this tool, we present guidelines for creating these types of interfaces in Chapter 6.

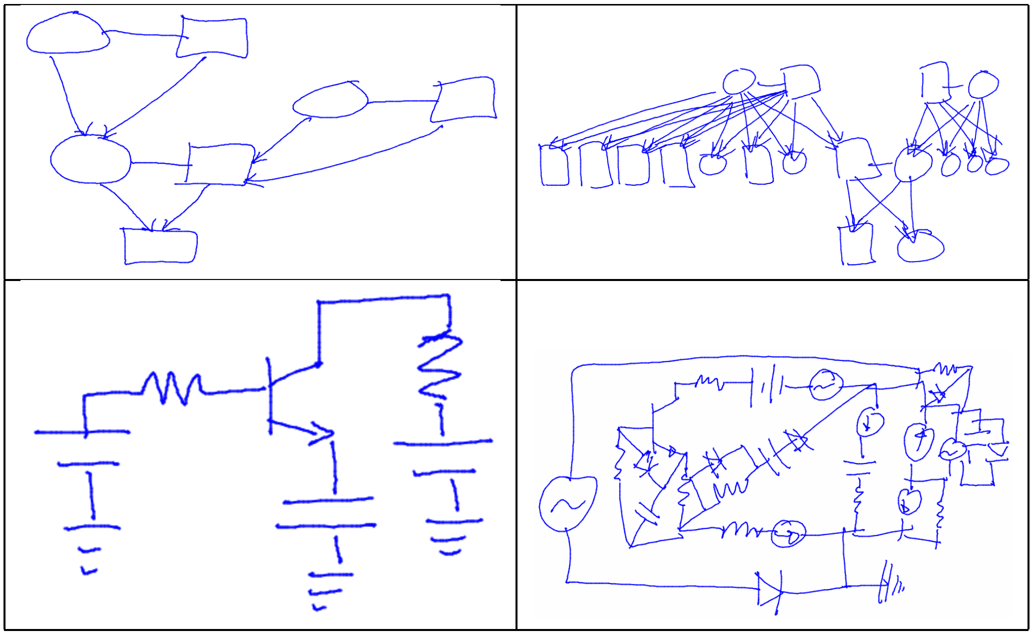


Figure 1-3: Examples of family trees and circuit diagram sketches.

1.5 Contributions

This work explores the task of online sketch understanding for early design. It makes five significant contributions within the fields of sketch understanding and intelligent user interface design. We present each contribution below in the order in which it will be discussed in this dissertation.

- First, this thesis presents a general framework for sketch recognition that can be extended to multiple domains. It presents a method for recognizing objects in sketches based on object descriptions so that the generic recognition engine can be extended to a new domain by providing structural descriptions for the objects in that domain; no training data is necessary, although training data may be used to determine probabilities used in the system. Furthermore, this system works with hierarchical descriptions that separate low-level shape descriptions from domain-specific shape patterns. This separation allows the engine to be extended to multiple domains without creating new shape recognizers for lower-level shapes.

- Second, this thesis presents a novel application of dynamically constructed Bayesian networks specifically developed for two-dimensional, constraint-based recognition. Hidden Markov Models (HMMs) assume one-dimensional (usually time-ordered) data, while Markov Random Fields (MRFs) generally allow for only local influence between neighboring pieces of the image or sketch. We specify Bayesian networks that can be dynamically constructed in response to the user's strokes. In contrast to HMMs and MRFs, these Bayesian networks can reflect arbitrary relationships between any pieces of the user's sketch.

- Third, our system simultaneously segments and recognizes the objects in freely-drawn, often messy sketches, making fewer assumptions about the way shapes will be drawn than previous recognition systems. We use domain information to effectively handle noise in the drawing by reinterpreting strokes that are misrecognized in the low-level step.

- Fourth, our implementation has been applied successfully to three domains: relationship diagrams, family tree diagrams and electrical engineering diagrams. There are few, if any, reported recognition results for our task. We collected real sketches of family tree diagrams and electrical engineering diagrams and present recognition results on these test sets. Using these results, we analyze the practical strengths of our approach and discuss remaining challenges for multi-domain sketch recognition systems.

- Finally, we used our recognition engine to explore the design implications of building sketch recognition user interfaces (SkRUIs). While pen-based user interfaces have been explored previously, our system enables the exploration of a fundamentally new type of interaction in which the user sketches freely while the system recognizes the drawing in the background. We implemented a sketch recognition-based interface for creating PowerPoint diagrams and performed informal user studies to explore the difficulties involved in creating this type of interface. We present guidelines for building SkRUIs, taking into account both user preferences and realistic requirements of sketch recognition algorithms.

1.6 Outline

The structure of this thesis is as follows. The next three chapters discuss our approach to sketch recognition: Chapter 2 describes how knowledge is represented in our system; Chapter 3 describes how uncertainty is managed during the recognition process; and Chapter 4 describes how our system searches for possible interpretations of the user's sketch, including how it recovers from low-level recognition errors. Next, Chapter 5 presents the application domains and the results of applying our engine to these domains. Chapter 6 presents our PowerPoint application and the results of our user study. Finally, Chapter 7 discusses related work in sketch understanding and similar recognition tasks, and Chapter 8 presents conclusions.

Chapter 2

Knowledge Representation

The problem this dissertation addresses is how to build a sketch recognition system that is easily extensible to a number of domains. Central to this problem is the question of how to represent the symbols to be recognized in a given domain.

The goal of any recognition system is to match its input against against an internal representation of a shape or set of shapes and identify the best match or matches (if any) for the given input. However, how each system represents the shape or shapes to be recognized (and consequently how each system matches the input to this internal representation) varies from system to system. For example, one system might represent each shape as a bit-mapped image template of the canonical form of that shape. Then to perform recognition, that system would apply a series of legal transformations to the input data (e.g., rotation, scaling) to determine whether or not the pixels in the input can be made to line up with the pixels in the template. In contrast, a different system might represent each shape not as an image but as a collection of features extracted from the shapes. Examples of potential features include the ratio between the height and width of the bounding box of the shape, the total length of the strokes in the shape relative to the size of the bounding box, the number of corners in the shape, etc. Recognition in this system would then extract the same features from the input data and determine whether or not the features extracted from the input data are close enough to the features stored for each shape.

While many different representations can be used to perform recognition, the choice of internal shape representation affects recognition task difficulty. In the example above, recognition using the feature-based approach is more straightforward than the template-matching approach as it involves only a relatively small number of easy to calculate features rather than multiple transformations of the whole input. However, depending on the shapes in the domain, it may be extremely difficult to devise a set of features that reliably separates one shape from another.

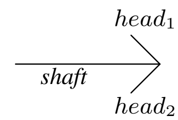
Our system represents symbols to be recognized using a probabilistic, hierarchical description language. In choosing our representation, we considered several desired functionalities of our recognition system. First, the system should be extensible to new domains, requiring few training examples. Second, the system should be able to distinguish between legal and illegal shape transformations when performing recognition. Legal transformations include not only rotation, translation and scaling but also some non-rigid shape transformations. For example, the angle between a line in the head of an arrow and the shaft may range from about 10 degrees to about 80 degrees, but an angle greater than 90 degrees is not acceptable. Third, we would like to use this recognition system to compare various techniques for providing recognition feedback to the user, so the system should be able to recognize the sketch as the user draws to allow the system to potentially provide recognition feedback at any point in the drawing process. Finally, the system should be able to cope with the noise inherent in hand-drawn diagrams (e.g., lines that are not really straight, corners that do not actually meet, etc.).

This chapter describes our hierarchical description language and discusses how this choice of representation allowed us to construct a system that meets the requirements above. We begin by introducing the deterministic properties of the language, then discuss how uncertainty is incorporated into the descriptions. Finally, we discuss the advantages and disadvantages of this choice of representation.

2.1 Hierarchical Shape Descriptions

Each shape in the domain to be recognized is described using a hierarchical description language, called LADDER, being developed by others in our group [37]. We introduce the language through examples from the family tree and circuit domains.

We refer to any pattern recognizable in a given domain as a *shape*. *Compound shapes* are those composed of *subshapes*. Compound shapes must be non-recursive. Describing a compound shape involves specifying its subshapes and any necessary *constraints* between those subshapes. As an example, the description of an arrow is given in Figure 2-1. The arrow has three subshapes—the line that is the shaft and the two lines that combine to make the head. The constraints specify the relative size, position and orientation necessary for these three lines to form an arrow shape



DEFINE ARROW
(**Subshapes**
(Line *shaft* $head_1$ $head_2$))
(**Constraints**
(coincident *shaft*.p_1 $head_1.p_1$)
(coincident *shaft*.p_1 $head_2.p_2$)
(equalLength $head_1$ $head_2$)
(larger *shaft* $head_1$)
(acuteAngle $head_1$ *shaft*)
(acuteAngle $head_2$ *shaft*))

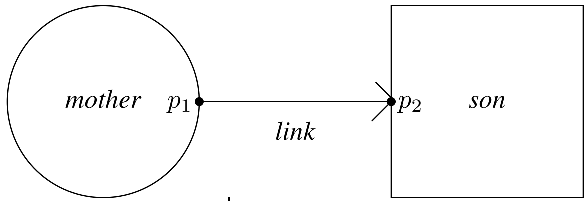
Figure 2-1: The description of the shape "arrow." Once defined, this shape can be used in descriptions of domain-specific shapes, as in Figure 2-2 and Figure 2-3.

(as opposed to just being three arbitrary lines). Once a shape has been defined, other shapes may use that shape in their descriptions. For example, the child-link symbol in the family tree domain (Figure 2-2) and the current source symbol in the circuit domain (Figure 2-3) both use an arrow as a subshape.

Shapes that cannot be broken down into subshapes are called *primitive shapes*. The set of primitive shapes includes free-form strokes, lines, arcs and ellipses. Although primitive shapes cannot be decomposed into subshapes, they may have named *subcomponents* that can be used when describing other shapes, e.g., the endpoints of a line, p_1 and p_2, used in Figure 2-1.

Domain shapes are shapes that have semantic meaning in a particular domain. Child-link and current-source are both domain shapes, but arrow and line are not because they are not specific to any one domain. *Domain patterns* are combinations of domain shapes that are likely to occur, for example the child-link pointing from a female to a male, indicating a relationship between mother and son in Figure 2-2. Compound shape descriptions with no constraints (e.g., the child-link description) are used to rename a generic geometric shape (e.g., the arrow) as a domain shape so that domain-specific semantics may be associated with the shape.

The language provides support for a shorthand vector notation to handle shapes that have many repeating components (e.g., the lines in a quadrilateral (Figure 2-4)). This shorthand is provided for convenience in writing shape descriptions. SketchREAD converts the components in the vector



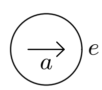
DEFINE FEMALE
(**Subshapes**
(Ellipse e))

DEFINE MALE
(**Subshapes**
(Quadrilateral q))

DEFINE CHILD-LINK
(**Subshapes**
(Arrow a))

DEFINE MOTHER-SON
(**Subshapes**
(Male *son*)
(Female *mother*)
(Child-Link *link*))
(**Constraints**
(touches *link.shaft.*p_1 *mother*)
(touches *link.shaft.*p_2 *son*))

Figure 2-2: Descriptions of several domain shapes (Female, Male and Child-Link) and one domain pattern (Mother-Son) in the family tree domain.



DEFINE CURRENT-SOURCE
(**Subshapes**
(Ellipse e)
(Arrow a))
(**Constraints**
(contains e a))

Figure 2-3: The description of a Current Source from the circuit domain.

into separately represented components. Appendix A gives a complete set of shape descriptions for each of our domains as well as the available constraints.

DEFINE QUADRILATERAL
(**Subshapes**
(vector Line $l[4]$))
(**Constraints**
(coincident $l[i].p_2$ $l[i+1].p_1$)
(coincident $l[4].p_2$ $l[1].p_1$))

Figure 2-4: The description of a quadrilateral written using compact vector notation.

2.2 Handling Noise in the Drawing

To recognize symbols, the system compares the user's input against the available shape descriptions. Recognition information for primitive shapes and constraints are built into SketchREAD; compound shapes are recognized by first identifying the subshapes and then verifying that the necessary constraints between those subshapes hold. The system's goal is to choose the best set of domain shapes for a given set of strokes.

While recognition appears straightforward, Chapter 1 illustrated that ambiguity in the drawing can make recognition more difficult. It is often difficult to recognize low-level shapes (e.g., lines are never perfectly straight) and constraints often do not hold exactly (e.g., lines are rarely precisely the same length). The next two chapters discuss the details of the recognition process. Here, we describe the language constructs that help the system cope with the inevitable noise and ambiguities in the drawing. We discuss two different types of variation supported by our representation: signal-level noise and description-level variation.

2.2.1 Signal-level noise: Objective vs. Subjective measures

Shape descriptions specify the subshapes and constraints needed to form a higher-level shape, and the system's goal is to determine whether or not the necessary shapes exist and the constraints between them hold. However, sketches often are messy, and people rarely draw constraints that hold exactly. For example, although a user intends to draw two parallel lines, it is unlikely that these lines will be exactly parallel. We call this type of variation *signal-level noise*.

Because of signal-level noise, low-level shape and constraint interpretations must be based both on the data and on the context in which that shape or constraint appears. Consider whether or not the user intended for the two bold lines in each drawing in Figure 2-5 to connect. In 2-5(a) and 2-5(b),

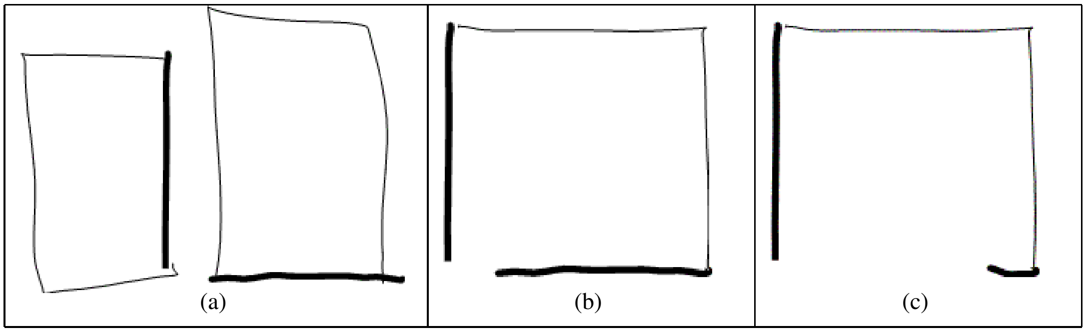


Figure 2-5: The importance of both data and context in determining whether or not the bold lines were intended to connect.

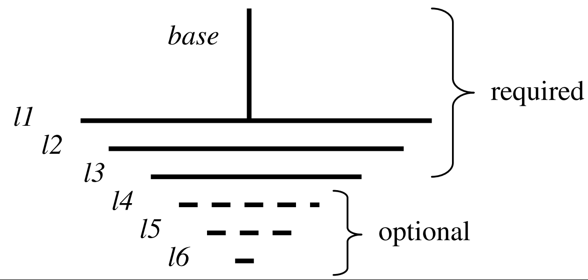
the bold lines are identically spaced, but the context surrounding them indicates that in 2-5(b) the user intended for them to connect, while in 2-5(a) the user did not. On the other hand, the stroke information should not be ignored. The thin lines in drawings 2-5(b) and 2-5(c) are identical, but the distance between the endpoints of the bold lines in these figures indicate that the these lines are intended to connect in 2-5(b) but not in 2-5(c).

For each low-level shape and constraint we identify an objectively measurable property that corresponds to that shape or constraint. For example, the property related to the constraint *coincident* is the distance between the two points in question normalized by the length of the lines containing the points in question. This objectively measurable property allows the system to separate the information provided by the stroke data from the information provided by the surrounding context to determine whether or not the constraint actually holds. Chapter 3 discusses precisely how these low-level measurements and the contextual data are combined, and Appendix B give further details.

2.2.2 Description-level variation: Optional Components and Constraints

All of the shapes considered so far have been modeled using a fixed number of subshapes and a set of required constraints between those subshapes. These descriptions signify that when a user draws these symbols, she should draw all of the subparts specified. In contrast, some shapes have subcomponents that can be omitted legally when they are drawn. For example, consider the ground symbol described in Figure 2-6. The user may draw up to six horizontal lines, but three of these lines optionally may be omitted. These three lines are flagged as *optional* in the shape description, indicating that when the user draws the ground symbol, the number of horizontal lines she includes may vary from three to six. We call this type of variation *description-level variation*.

Constraints may also be flagged as optional, indicating that they often hold, but are not required



Standard Notation	**Vector Notation**
DEFINE GROUND (**Subshapes** (Line *base*) (Line l_1) (Line l_2) (Line l_3) (*optional* Line l_4) (*optional* Line l_5) (*optional* Line l_6)) (**Constraints** (perpendicular *base* l_1) (touches l_1 $base.p_2$) (parallel l_1 l_2) (parallel l_2 l_3) (parallel l_3 l_4) (parallel l_4 l_5) (parallel l_5 l_6) (smaller l_2 l_1) (smaller l_3 l_2) (smaller l_4 l_3) (smaller l_5 l_4) (smaller l_6 l_5) (nextTo l_1 l_2) (nextTo l_2 l_3) (nextTo l_3 l_4) (nextTo l_4 l_5) (nextTo l_5 l_6) (aligned l_1 l_2) (aligned l_2 l_3) (aligned l_3 l_4) (aligned l_4 l_5) (aligned l_5 l_6))	DEFINE GROUND (**Subshapes** (Line *base*) (vector Line $l[3, 6]$)) (**Constraints** (perpendicular *base* l_1) (touches l_1 *base*.p_2) (parallel $l[i]$ $l[i+1]$) (smaller $l[i+1]$ $l[i]$) (nextTo $l[i]$ $l[i+1]$) (aligned $l[i]$ $l[i+1]$))

Figure 2-6: The ground symbol from the Circuit Diagram domain. Three of the lines in the symbol are optional, meaning the user may draw a valid ground symbol that does not contain those lines. Both standard and vector descriptions are given for this symbol.

in the description of a symbol. For example, we could define the domain shape `wire` as having the single subshape *line* and the optional constraint that the line is horizontal or vertical. This constraint is not strictly required, as wires may be drawn diagonally, but they are often drawn either horizontally or vertically. Constraints pertaining to optional components are considered required if the optional component is present unless they are explicitly flagged as optional.

Optional components may also be specified using the shorthand vector notation described above. To specify a vector of components, some of which are optional, a minimum and a maximum number of components must be specified. Components up to the minimum number specified are required, those between the minimum and maximum number are optional. The vector specification for the ground symbol is given in Figure 2-6.

Understanding the difference between signal-level noise and description-level variation is central to understanding our representation of uncertainty. Signal-level noise is distinguished from description-level variation by considering the user's intent when she draws a symbol. For example, in a ground symbol the *base* line should be perpendicular to line l_1. In a given drawing, the angle between those lines may actually be far from 90 degrees due to signal-level noise (which might be caused by the user's sloppiness), but the lines are still intended to be perpendicular. On the other hand, the ground symbol may not contain line l_6, not because the user was being sloppy, but because the user did not intend to include it when drawing the symbol. We discuss how we model each variation in the next chapter.

2.3 Strengths and Limitations

We chose this symbolic, hierarchical representation based on the recognition system guidelines presented in the first part of this chapter. Here, we consider how this representation supports the creation of such a system. As every representation choice has trade-offs, we also consider the limitations of this approach and briefly discuss how these limitations can be addressed.

The first requirement was that our system must be extensible to new domains, requiring few training examples. To extend the system to a new domain, a user must simply describe the domain shapes and patterns for the new domain. Because the system can use the same hierarchical recognition process, it does not need to be trained with a large number of examples for each new shape. Furthermore, basic geometric shapes can be defined once and reused in a number of domains.

The second requirement was that our system must accept legal non-rigid transformations of

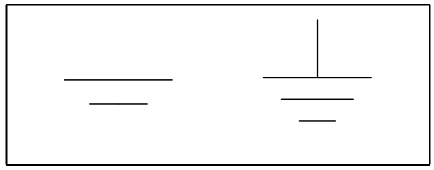


Figure 2-7: Battery (left) and ground (right) symbols from the Circuit Diagram domain. Note that the battery symbol is a subset of the ground symbol.

sketched symbols without accepting illegal transformations. This requirement is handled by the fact that constraints can be defined to accept a wide range of relationships between shapes. For example, *acuteAngle* refers to any angle less than 90 degrees. Furthermore, only constraints explicitly stated in the shape definition are verified. Constraints that are not specified may vary without affecting the system's interpretation of the shape. For example, the definition of a quadrilateral given in Figure 2-4 does not constrain the relative lengths of the lines or the angles between those lines, leaving the system free to interpret any set of four correctly connected lines as a quadrilateral.

The third requirement was that the system be able to recognize a sketch as it is being drawn. To support this goal, our representation in terms of a shape's subcomponents allows the system to detect when it has seen only some of the subcomponents of a given shape. Using these partial interpretations, the system can decide which interpretations are likely complete and which might still be in the process of being drawn. This capability is particularly important when shape descriptions overlap, as in the battery and the ground symbol in the circuit domain (Figure 2-7). When the user draws what could look like a battery or part of a ground symbol, the system can detect the partial ground interpretation and wait until the user has drawn more strokes to give its final interpretation instead of immediately interpreting the strokes as a battery.

The final requirement was that our system must deal with the noise in hand-drawn diagrams. Our representation allows us to handle signal-level noise by separating low-level objective measurements from judgments about whether or not constraints hold.

Although our representation satisfies the above requirements, it also imposes some restrictions. First, even with a well designed language, specifying shape descriptions may be difficult or time-consuming. To address this difficulty, other members of our group are developing a system to learn shape descriptions based on few examples [76]. As the user draws a shape, the learning system parses the user's strokes into low level components such as lines, arcs, and ellipses. The learner then calculates the existing constraints between these components and uses perceptual cues to deduce which constraints are most important the shape description. Once the system has learned a shape (e.g., a rectangle) it can then use that shape in its description of other shapes (e.g., a house). The output of the learning system is a description of a domain shape in the visual language.

Second, even if we could build a system to learn shape descriptions, some shapes may be difficult or impossible to describe in terms of any simple low-level components or constraints. Those domains with free-form shapes, such as architecture, may have many shapes that cannot be easily described. Our representation is appropriate only for domains with highly structured symbols.

Finally, using this representation it is difficult to represent text or unrecognized strokes. This limitation must be addressed in the recognition system itself. The system should be capable of detecting text or unrecognized strokes and processing them using a different recognition technique or leaving them as unrecognized. Separating text from diagrams is a challenging problem that we do not address. A complementary system must be used in conjunction with our recognition system to distinguish between annotations and diagrams.

Chapter 3

Hypothesis Evaluation

The problem of two-dimensional sketch recognition is to parse the user's strokes according to the specified visual language to determine the best set of known patterns to describe the input. In order for a shape to be recognized, the system must detect the necessary subshapes and constraints between those subshapes. This two-dimensional parsing problem presents a challenge for a real-time system. Because sketched objects rarely appear in their canonical representations (for example, an arrow may point in any direction, may vary in size, and may be drawn in a variety of styles), recognizing an object in a sketch involves recovering the underlying shape of the object in the face of this legal variation. Low-level interpretations potentially can help guide the search for possible higher-level interpretations; for example, if the system detects two connected lines, it can first try to match a quadrilateral whose corner lines up with the connection between the lines. However, noise in the input makes it impossible for the system to recognize low-level shapes with certainty or to be sure whether or not constraints hold. Low-level misinterpretations cause higher-level interpretations to fail as well. Trying all possible interpretations of the user's strokes guarantees that an interpretation will not be missed, but it is infeasible due to the exponential number of possible interpretations.

To solve this problem we use a combined bottom-up and top-down recognition algorithm that generates the most likely interpretations first (bottom-up), then actively seeks out parts of those interpretations that are still missing (top-down). Our approach uses a novel application of dynamically constructed Bayesian networks to evaluate partial interpretation hypotheses and then expands the hypothesis space by exploring the most likely interpretations first. The system does not have to try all combinations of all interpretations, but can focus on those interpretations that contain at least a subset of easily-recognizable subshapes and can recover any low-level subshapes that may have been mis-recognized.

We use a two-stage generate-and-test method to explore possible interpretations for the user's strokes. In the first stage, the system generates a number of *hypotheses*, or possible interpretations for the user's strokes, based on the shape descriptions described in Chapter 2. We refer to each shape description as a *template* with one *slot* for each subpart. A *shape hypothesis* is a template with an associated mapping between slots and strokes. Similarly, a *constraint hypothesis* is a proposed constraint on one or more of the user's strokes. A *partial hypothesis* is a hypothesis in which one or more slots are not bound to strokes. Our method of exploring the space of possible interpretations depends on our ability to assess both complete and partial hypotheses for the user's strokes. This chapter describes our hypothesis evaluation technique; the next chapter describes how these hypotheses are generated.

We use a constraint-based approach to hypothesis evaluation, evaluating higher-level shapes and patterns by evaluating their subcomponents and the constraints between them. Our method for evaluating possible interpretations differs from previous constraint-based approaches (e.g., [26, 31, 78]) in two ways. First, our technique must be able to evaluate partial hypotheses (e.g., an arrow with no shaft) because we wish to interpret drawings as they develop, and because the strength of partial hypotheses guides the interpretation of new strokes as they are processed. Second, because sketches are noisy, we cannot determine whether or not constraints hold simply by looking at the stroke data. For example, two lines intended to connect to form a corner of a square may not actually be connected, and there is no threshold we use to determine how far apart they may be and still be considered connected. The decision as to whether they should be considered connected depends on context (see Figure 2-5). The system's certainty in an interpretation should be influenced both by the stroke data and by the context provided by the surrounding possible interpretations.

Bayesian networks provide a way of reasoning about uncertainty in the world by combining prior knowledge and observations of the world. To handle both of the above issues, we use a Bayesian network in which each node represents a hypothesis for part of the user's drawing. Missing data can be treated as unobserved nodes in the network when the system assesses likely hypotheses for the strokes that have been observed thus far. Furthermore, the system's belief in a given hypothesis can be influenced both by the stroke data (through the node's children) and the context in which those shapes appear (through the node's parents). This chapter describes how we apply dynamically constructed Bayesian networks to the task of constraint-based recognition.

3.1 Dynamically Constructed Graphical Models

Time-based graphical models, including Hidden Markov Models (HMMs) and Dynamic Bayesian Networks (DBNs), have been applied successfully to time-series data in tasks such as speech understanding. To the extent that stroke order is predictable, HMMs and DBNs may be applied to sketch understanding. Ultimately, however, sketch understanding is different because we must model shapes based on two-dimensional constraints (e.g., intersects, touches) rather than on temporal constraints (i.e., follows), and because our models cannot simply unroll in time as data arrives (we cannot necessarily predict the order in which the user will draw the strokes, and things drawn previously can be changed). Therefore, our network represents spatial relationships rather than temporal relationships.

It is not difficult to use Bayesian networks to model spatial relationships. The difficult part of using Bayesian networks for sketch understanding is that they are traditionally used to model static domains in which the variables and relationships between those variables are known in advance. Static networks are not suitable for the task of sketch recognition because we cannot predict *a priori* the number of strokes or symbols the user will draw in a given sketch. In fact, there are many tasks in which the possible number of objects and relationships may not be modeled *a priori*. For example, when reasoning about military activity, the number of military units and their locations cannot be known in advance. For such tasks, models to reason about specific problem instances (e.g., a particular sketch or a particular military confrontation) must be dynamically constructed in response to a given input. This problem is known as the task of *knowledge-based model construction* (KBMC).

A number of researchers have proposed models for the dynamic creation of Bayesian networks for KBMC. Early approaches focused on generating Bayesian networks from probabilistic knowledge bases [29, 30, 34, 67]. A recently proposed representation representation, called Network Fragments, represents generic template knowledge directly as Bayesian network fragments that can be instantiated and linked together at run-time [51]. Finally, Koller *et al.* have developed a number of a number of object-oriented frameworks including Object Oriented Bayesian Networks (OOBNs) [46, 66] and Probabilistic Relational Models (PRMs) [28]. These models represent knowledge in terms of relationships among objects and can be instantiated dynamically in response to the number of objects in a particular situation.

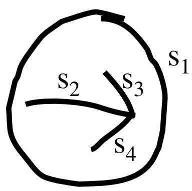
Although the above frameworks are powerful, they are not directly suitable for sketch recognition because they are too general in some respects and too specialized in others. First, with this type of general model, it is a challenge simply to decide how to frame our recognition task in terms of objects, network fragments, or logical statements. Second, because these models are general, they do not make any assumptions about how the network will be instantiated. Because of the size of the networks potentially generated for our task, it is sometimes desirable to generate only part of a complete network, or to prune nodes from the network. In reasoning about nodes that are in the network, we must account for the fact that the network may not be fully generated or relevant information may have been pruned from the network. Finally, these models are too specific in that they have been optimized for responding to specific queries, for example, "What is the probability that a particular battery in our military force has been hit?" In contrast, our model must provide probabilities for a full set of possible interpretations of the user's strokes.

We present a framework for dynamically constructing Bayesian networks specifically targeted to the task of constraint-based recognition. Our framework is closely related to both Network Fragments and Object Oriented Bayesian Networks, but designed to handle the specific problems presented above that arise in the recognition task. We provide a Bayesian network representation to perform constraint-based recognition. We show how these networks grow in response to the incoming data and how to represent information compactly to reduce the size and complexity of the network. We begin by considering how the system evaluates a single hypothesis for a given set of strokes, and then consider how to aggregate individual hypotheses to determine the best set of interpretations for a complete sketch. Finally, we give implementation details and discuss our inference method.

3.2 Shape Fragments: Evaluating a Single Hypothesis

Briefly, Bayesian networks consist of two parts: a Directed Acyclic Graph that encodes *which* factors influence one another, and a set of Conditional Probability Distributions which specify *how* these factors influence one another.[1] Each node in the graph represents something to be measured, and a link between two nodes indicates that the value of one node is directly dependent on the value of the other. Each node contains a conditional probability function (CPF), represented as a conditional probability table (CPT) for discrete variables, specifying how it is influenced by its

[1]We provide enough background on Bayesian networks to give the reader a high-level understanding of our model. To understand the details, those unfamiliar with Bayesian networks are referred to [12] for an intuitive introduction and [44] for more details.



<u>CS hypothesis *CS₁*</u>

Subshapes*:*

e ellipse hypothesis E_1

a arrow hypothesis A_1

Constraints:

C_1 = (contains `ellipse-fit`(s_1)
`shape-fit`(s_2, s_3, s4))

Arrow hypothesis A_1

Subshapes*:*

shaft line hypothesis L_1

head₁ line hypothesis L_2

head₂ line hypothesis L_3

Constraints:

C_2 = (coincident `line-fit`(s_2).p1 `line-fit`(s_3).p1)
C_3 = (coincident `line-fit`(s_2).p1 `line-fit`(s_4).p1)
C_4 = (equalLength `line-fit`(s_3) `line-fit`(s4))
C_5 = (shorter `line-fit`(s_3) `line-fit`(s_2))
C_6 = (acuteAngle `line-fit`(s_2) `line-fit`(s_3))
C_7 = (acuteAngle `line-fit`(s_2) `line-fit`(s_4))

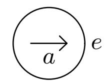
Primitive hypotheses

E_1 = `ellipse-fit`(s_1)
L_1 = `line-fit`(s_2)
L_2 = `line-fit`(s_3)
L_3 = `line-fit`(s_4)

Figure 3-1: A single current source hypothesis (CS_1) and associated lower-level hypotheses. Shape descriptions for the arrow and current source (with labeled subshapes) are given in Figures 3-2 and 3-3.

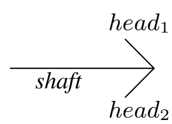
parents.

To introduce our Bayesian network model, we begin by considering how to evaluate the strength of a single current source (CS) hypothesis for the stokes in Figure 3-1. The description of a current source symbol is given Figure 3-2. Based on the hierarchical nature of the shape descriptions, we use a hierarchical method of hypothesis evaluation. Determining the strength of a particular current source hypothesis is a matter of determining the strengths of its corresponding lower-level shape and constraint hypotheses. A particular current source hypothesis, CS_1, specifies a mapping between the subparts in the current source description and the user's strokes via lower-level hypotheses for the user's strokes (Figure 3-1). E_1 is an ellipse hypothesis for s_1, A_1 is an arrow hypothesis involving strokes s_2, s_3 and s_4 (through its line hypotheses), and C_1 a constraint hypothesis that an ellipse fit for stroke s_1 contains strokes s_2, s_3, and s_4. A_1 is further broken down into three line hypotheses (L_1, L_2 and L_3) and six constraint hypotheses ($C_2, ..., C_7$) according to the description of the arrow (Figure 3-3) Thus, determining the strength of hypothesis CS_1 can be transformed into the problem of determining the strength of a number of lower-level shape and constraint hypotheses.



DEFINE CURRENT-SOURCE
(**Subshapes**
  (Ellipse e)
  (Arrow a))
(**Constraints**
  (contains e a))

Figure 3-2: The description of a current source from the circuit domain.



(**Subshapes**
  (Line *shaft*)
  (Line $head_1$)
  (Line $head_2$)
(**Constraints**
  (coincident *shaft*.p_1 $head_1.p_1$)
  (coincident *shaft*.p_1 $head_2.p_2$)
  (equalLength $head_1$ $head_2$)
  (shorter $head_1$ *shaft*)
  (acuteAngle $head_1$ *shaft*)
  (acuteAngle $head_2$ *shaft*))

Figure 3-3: The description of an arrow.

3.2.1 Network Structure

The Bayesian network for this recognition task is shown in Figure 3-4. There is one node in the network for each hypothesis described above, and each of these nodes represents a boolean random variable that reflects whether or not the corresponding hypothesis is correct. The nodes labeled $O_1, ..., O_{11}$ represent measurements of the stroke data that correspond to the constraint or shape to which they are linked. The variables corresponding to these nodes have positive real numbered values. For example, the variable O_2 is a measurement of the squared error between the stroke s_1 and the best fit ellipse to that stroke. The value of O_2 is a real number between 0 and the maximum

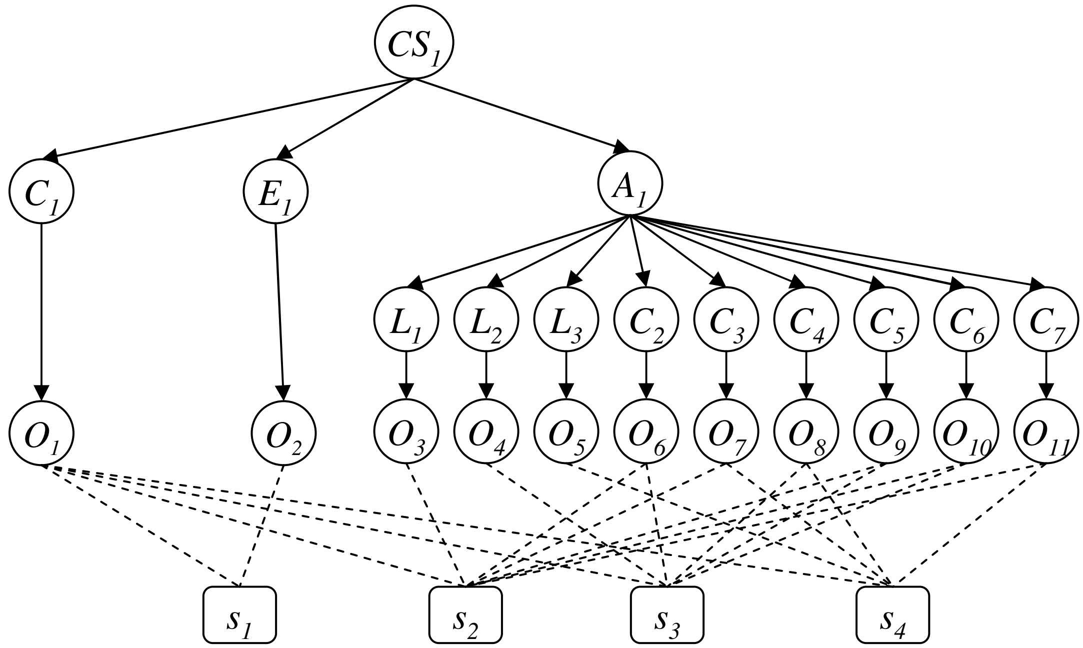


Figure 3-4: A Bayesian network to verify a single current source hypothesis. Labels come from Figure 3-1.

possible error between any stroke and an ellipse fit to that stroke. The boxes labeled $s_1, ..., s_4$ are not part of the Bayesian network but serve to indicate the stroke or strokes from which each measurement, O_i, is taken (e.g., O_2 is measured from s_1). $P(CS_1 = t|ev)$ (or simply $P(CS_1|ev)$)[2], where ev is the evidence observed from the user's strokes, represents the probability that the hypothesis CS_1 is correct.

There are three important reasons why the links are directed from higher-level shapes to lower-level shapes instead of in the opposite direction. First, whether or not a higher-level hypothesis is true directly influences whether or not a lower-level hypothesis is true. For example, if the arrow hypothesis A_1 is true, then it is extremely likely that all three line hypotheses, L_1, L_2, L_3, are also true. Second, this representation allows us to model lower-level hypothesis as conditionally independent given their parents, which reduces the complexity of the data needed to construct the network. Finally, continuous valued variables are difficult to incorporate into a Bayesian network if they have discrete valued children. Our representation ensures that the measurement nodes, which have continuous values, will be leaf nodes. These nodes can be pruned when they do not have evidence, thus simplifying the inference process.

Each shape description constrains its subshapes only relative to one another. For example, an

[2]Throughout this chapter, t means true, and f means false.

arrow may be made from *any* three lines that satisfy the necessary constraints. Based on this observation, our representation models a symbol's subshapes separately from the necessary constraints between those subshapes. For example, node L_1 represents the hypothesis that stroke s_2 is a line. It's value will be true if the user intended for s_2 to be any line, regardless of its position, size or orientation. Similarly, C_2 represents the hypothesis that the line fit to s_2 and the line fit to s_3 are coincident.

The conditional independence between subshapes and constraints might seem a bit strange at first. For example, whether or not two lines are the same length seems to depend on the fact that they are lines. However, observation nodes for constraints are calculated in such a way that their value is not dependent on the true interpretation for a stroke. For example, when calculating whether or not two lines are parallel, which involves calculating the different in angle between the two lines, we first fit lines to the strokes (regardless of whether or not they actually look like lines), then measure the relative orientation of those lines. How well these lines fit the strokes is not considered in this calculation.

The fact that the shape nodes are not directly connected to the constraint nodes has an important implication for using this model to perform recognition: There is no guarantee in this Bayesian network that the constraints will be measured from the correct subshapes because the model allows subshapes and constraints to be detected independently. For example, C_3 in Figure 3-4 indicates that L_2 and L_3 (the two lines in the head of an arrow) must be the same length, not simply that any two lines must be the same length. To satisfy this requirement, The system must ensure that O_6 is measured from the same strokes that O_3 and O_4 were measured from. We use a separate mechanism to ensure that only legal bindings are created between strokes and observation nodes.

The way we model shape and constraint information has two important advantages for recognition. First, this Bayesian network model can be applied to recognize a shape in any size, position and orientation. CS_1 represents the hypothesis that $s_1, ..., s_4$ form a current source symbol, but the exact position, orientation and size of that symbol is determined directly from the stroke data. To consider a new hypothesis for the user's strokes, the system simply creates a copy of the necessary Bayesian network structure whose nodes represent the new hypotheses and whose measurement nodes are linked to a different set of the user's strokes. Second, the system can allow competing higher-level hypotheses for a lower-level shape hypothesis to influence one another by creating a network in which two or more hypotheses point to the same lower-level shape node. For example, the system may consider an arrow hypothesis and a quadrilateral hypothesis involving the same line hypothesis for one of the user's strokes. Because the line hypothesis does not include any higher-level shape-specific constraint information, both an arrow hypothesis node and a quadrilateral hypothesis node can point to the single line hypothesis node. These two hypotheses then become alternate, competing explanations for the line hypothesis. We further discuss how hypotheses are combined below.

Our model is generative, in that we can use the Bayesian network for generation of values for the nodes in the network based on the probabilities in the model. However, our model is fundamentally different from the standard generative approach used in computer vision in which the system generates candidate shapes (for example, a rightward facing arrow) and then compares these shapes to the data in the image. The difference is that the lowest level of our network represents measurements of the strokes, not actual stroke data. So although our model can be used to generate values of stroke data measurements, it cannot be used to generate shapes which can be directly compared to the user's strokes. However, because the system can always take measurements from existing stroke data, our model is well suited for hypothesis evaluation.

3.2.2 Conditional Probability Distributions

Next, we consider the intuition behind the CPTs for a node given its parents for the hypotheses in Figure 3-1. We begin by considering the distribution $P(E_1|CS_1)$. Intuitively, we set $P(E_1 = t|CS_1 = t) = 1$ (and conversely, $P(E_1 = f|CS_1 = t) = 0$), meaning that if the user intended to draw CS_1, she certainly intended to draw E_1. This reasoning follows from the fact that the ellipse is a required component of the CS symbol (and that the user knows how to draw CS symbols). $P(E_1|CS_1 = f)$, on the other hand, is a little less obvious. Intuitively, it represents the probability that the user intended to draw E_1 even though she did not intend to draw CS_1. This probability will depend on the frequency of ellipses in other symbols in the domain (i.e., higher if ellipses are common).

Because CS_1 has no parents, it must be assigned a prior probability. This probability is simply how likely it is that the user will draw CS_1. This probability will be high if there are few other shapes in the domain or if CS symbols are particularly prominent, and low if there are many symbols or if the CS symbol is rare. Exactly how these prior probabilities are determined is discussed further in Chapter 5.

The bottom layer of the network accounts for signal level noise by modeling the differences between the user's intentions and the strokes that she draws. For example, even if the user intends to draw L_1, her stroke likely will not match L_1 exactly, so the model must account for this variation. Consider $P(O_2|E_1 = t)$. If the user always drew perfect ellipses, this distribution would be 1 when $O_2 = 0$, and 0 otherwise. However, most people do not draw perfect ellipses (due to inaccurate pen and muscle movements), and this distribution allows for this error. It should be high when O_2 is close to zero, and fall off as O_2 gets larger. The wider the distribution, the more error the system will tolerate, but the less information a perfect ellipse will provide.

The other distribution needed is $P(O_2|E_1 = f)$ which is the probability distribution over ellipse error given that the user did not intend to draw an ellipse. This distribution should be close to uniform, with a dip around 0, indicating that if the user specifically does not intend to draw an ellipse, she might draw any other shape, but probably won't draw anything that resembles an ellipse. Details about how we determined the conditional probability distributions between primitive shapes and constraints and their corresponding measurement nodes are given in Appendix B.

3.2.3 Observing Evidence from Stroke Data

Finally, we discuss how information from the user's strokes is incorporated into the network to influence the system's belief in CS_1. If we assume that the user is done drawing, the values of $O_1, ..., O_{11}$ are fully observable by taking measurements of the strokes. The system can then use those values to infer $P(CS_1|O_1, ..., O_{11})$. If we do not assume the user is done drawing, we may still evaluate $P(CS_1|ev)$ where the set ev contains all O_i corresponding to strokes the user has drawn so far.

We may model the current source (partial) hypothesis CS_1 even before the drawing is complete. An observation node that does not have an observed value intuitively corresponds to a stroke that the user has not yet drawn. Because observation nodes are always leaf nodes, the missing data has neither a positive nor a negative effect on the system's belief in a given interpretation. CS_1 may be strongly believed even if O_1 is missing. As described in Chapter 4, the system uses the strength of incomplete interpretations to help guide the search for missed low-level interpretations.

3.3 Recognizing a Complete Sketch

The Bayesian network introduced above can be used to detect a single instance of a CS symbol in any size, position or orientation. However, a typical sketch contains several different symbols as well as several instances of the same symbol.

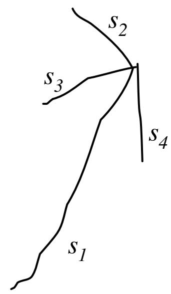


Figure 3-5: Four strokes, three of which form an arrow. The system might try both s_2 and s_3 as a line in the head of the arrow.

To detect other shapes in our domain, we may create a Bayesian network similar to the network above for each shape. We call each of these Bayesian networks a *shape fragment* because these fragments can be combined to create a complete Bayesian network for evaluating the whole sketch.

Above, we assumed that we were given a mapping between the user's strokes and the observation nodes in our network. In fact, the system must often evaluate a number of potential mappings between strokes and observation nodes. For example, if the user draws the four strokes in Figure 3-5, the system might try mapping both s_2 and s_3 to L_2. As described above, each interpretation for a specific mapping between strokes and observation nodes is called a hypothesis, and each hypothesis corresponds to a single node in the Bayesian network. In this section we discuss how multiple hypotheses are combined to evaluate a complete sketch.

Given a set of hypotheses for the user's strokes, the system instantiates the corresponding shape fragments and links them together to form a complete Bayesian network, which we call the *interpretation network*. To illustrate this process, we consider a piece of a network generated in response to Strokes 6 and 7 in the example given in Figure 1-1, which is reproduced in Figure 3-6. Figure 3-7 shows the part of the Bayesian network representing the possible interpretations that the system generated for these strokes. Each node represents a hypothesized interpretation for some piece of the sketch. For example, Q_1 represents the system's hypothesis that the user intended to draw a quadrilateral with strokes 6 and 7. A higher-level hypothesis is compatible with the lower-level hypotheses it points to. For example, if M_1 (the hypothesis that the user intended to draw a male with strokes 6 and 7) is correct, Q_1 (the hypothesis that the user intended to draw a quadrilateral with strokes 6 and 7) and $L_1, ..., L_4$ (the hypotheses that the user intended to draw 4 lines with strokes 6 and 7) will also be correct. Two hypotheses that both point to the same lower-level hypothesis represent competing interpretations for the lower level shape and are incompatible. For example, A_1, Q_1 are two possible higher-level interpretations for line L_1, only one of which may be true.

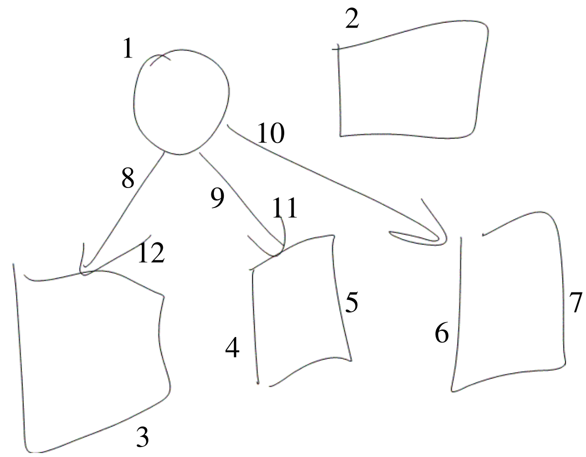


Figure 3-6: The partial sketch of a family tree from Chapter 1.

Each observation node is linked to a corresponding stroke or set of strokes. In a partial hypothesis, not all measurement nodes will be linked to stroke data. For example, A_1 is a partial hypothesis—it represents the hypothesis that L_1 and L_2 (and, hence, Stroke 6) are part of an arrow whose other line has not yet been drawn. Line nodes representing lines that have not been drawn (L_5 and L_6) are not linked to observation nodes because there is no stoke from which to measure these observations. We refer to these nodes (and their corresponding hypotheses) as *virtual*.

The probability of each interpretation is influenced both by stroke data (through its children) and by the context in which it appears (through its parents), allowing the system to handle noise in the drawing. For example, there is a gap between the lines in the top-left corner in Q_1 (see Figure 3-6); stroke data only weakly supports the corresponding constraint hypothesis (not shown individually). However, the lines that form Q_1 are fairly straight, raising probabilities of $L_1, ..., L_4$, which in turn raise the probability of Q_1. Q_1 provides a context in which to evaluate the coincident constraint, and because Q_1 is well supported by $L_1, ..., L_4$ (and by the other constraint nodes), it raises the probability of the coincident constraint corresponding to Q_1's top-left corner.

The fact that partial interpretations have probabilities allows the system to assess the likelihood of incomplete interpretations based on the evidence it has seen so far. In fact, even virtual nodes have probabilities, corresponding to the probability that the user (eventually) intends to draw these shapes but either has not yet drawn this part of the diagram or the correct low-level hypotheses have not yet been proposed because of low-level recognition errors. As we describe below, a partial interpretation with a high probability cues the system to examine the sketch for possible missed low-level interpretations.

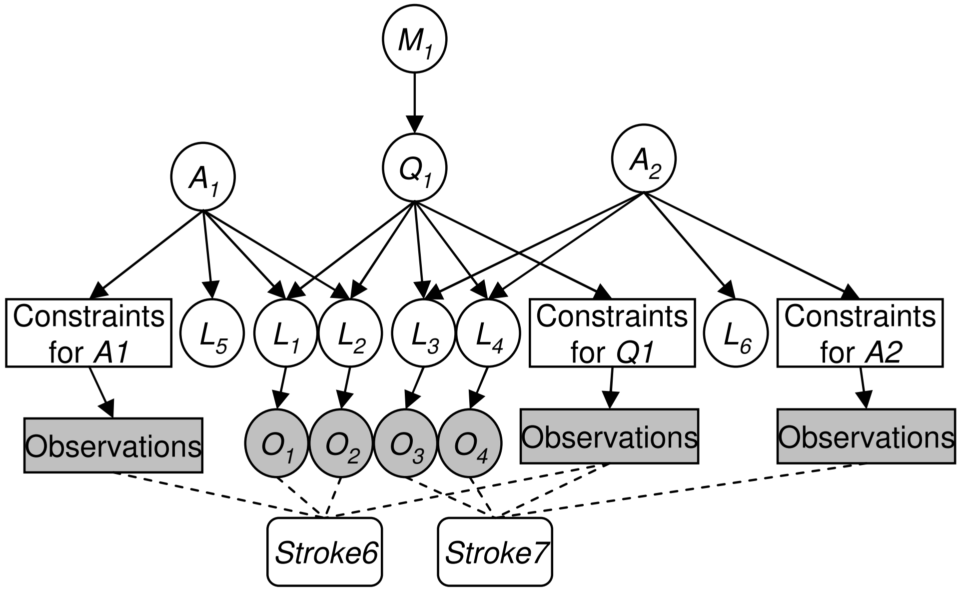


Figure 3-7: A portion of the interpretation network generated while recognizing the sketch in Figure 3-6.

3.3.1 Linking Shape Fragments

When Bayesian network fragments are linked during recognition, each node H_n may have several parents, $S_1 \ldots S_m$, where each parent represents a possible higher-level interpretation for H_n. We use a noisy-OR function to combine the influences of all the parents of H_n to produce the complete CPT for $P(H_n|S_1, \ldots, S_m)$. The noisy-OR function models the assumption that each parent can independently cause the child to be observed. For example, a single stroke might be part of a quadrilateral or an arrow, but both interpretations would favor that interpretation of the stroke as a line.

The intuition behind Noisy-OR is that each parent that is true will cause the child to also be true unless something prevents it from doing so. The probability that something will prevent a parent $S_i = t$ from causing $H_n = t$ to be true is $q_i = P(H_n = f|S_i = t)$. Noisy-OR assumes that all the q_i's are independent, resulting in the following:

$$P(H_n = t|S_1, S_2, \ldots, S_m) = 1 - \prod_i q_i$$

for each $S_i = t$. We set $q_j = P(H_n = f|S_j = t) = 0$ for all parents S_j in which H_n is a required subshape or constraint, and we set $q_k = P(H_n = f|S_k = t) = 0.5$ for all parents S_k in which H_n is an optional subshape or constraint. A consequence of these values is that $S_j = t \Rightarrow P(H_n|S_1, \ldots, S_m) = 1$ for any S_j in which H_n is required, which is exactly what we intended.

Noisy-OR requires that $P(H_n|S_1 = S_2, \ldots, S_m = f]) = 0$. The result of this requirement is that any shape or constraint has zero probability of appearing if it is not part of a higher-level shape or pattern. This behavior may be what is desired; however, if it is not, we may create an additional parent, S_a, to model the probability that the user intends to draw H_n alone, not as part of a shape or pattern.

We experimented with a noisy-XOR construct, implemented using a "gate node" similar to that described in [9]. Although this XOR construct is technically a better model for the semantics of sketch recognition (only one higher-level interpretation may be true), in practice we found that the noisy-OR semantics were simpler and in fact produced better results. Using the XOR construct, when one parent interpretation received slightly more support than the others, it immediately drove the probability of all the other parent interpretations to zero. In effect, a noisy-OR node in a Bayesian network behaves as a non-aggressive XOR. The behavior of Bayesian networks is such that if a node with evidence has multiple parents and one parent node receives high probability, it becomes the "cause" for the child node and "explains away" the need for the other parents. The fact that one parent is true does not actively prohibit the other parents from being true, but it causes their probabilities to tend back to their prior values. For more information, see [12].

3.3.2 Missing Nodes

Throughout this process, we have assumed that all of the hypothesized interpretations will exist as a node in the Bayesian network. However, for reasons discussed in Chapter 4, there are two reasons a hypothesis might be missing from the network. First, the system does not always initially generate all higher-level interpretations for a shape. Second, the system prunes unlikely hypotheses from the network to control the network's size.

We would like hypotheses that have not yet been generated or that have been pruned nevertheless to influence the strength of the hypotheses in the network. For example, if there are two potential interpretations for a stroke—a line and an arc—and the system prunes the line interpretation because it is too unlikely, the probability of the arc should go up. On the other hand, if the system has only generated an arc interpretation, and has not yet considered a line interpretation, the probability of the arc should remain modest because the stroke might still be a line.

We model nodes not present in the network through an additional parent, S_{np}, for each node H_n in the graph. We define $q_{np} = P(H_n = f|S_{np} = t)$ and set $S_{np} = t$. The value of q_{np} takes into account which nodes have been eliminated from the graph and which have not (yet) been instantiated, and it is calculated as follows. Let $T_1, \ldots, T_p$ be the set of shapes that have an element of type H_n as a child but do not exist as parents of H_n in the network. We refer to $T_1, \ldots, T_p$ as *potential parents* of H_n. For example, for node L_1 in Figure 3-7, this set would contain the single element ML because the marriage-link is the only shape in the family tree domain that has a line as a subshape but is not already a parent of L_1 in the graph. Let $T_1, \ldots, T_i$ be the subset of potential parents that have never appeared as a parent for H_n, and $T_{i+1}, \ldots, T_p$ be the subset that were once parents for H_n but have been pruned from the network. Then, $q_{np} = \prod_{j=1}^{i} 1 - P(T_j)$. We call $P(T_j)$ the *simple marginal probability* of T_j. It is calculated by calculating the marginal probability based only on the priors of the parents and ancestors of T_j in a network containing exactly one instance of each parent of T_j.

The effect of q_{np} is to allow only those shapes that have not yet been instantiated as parents of H_n to contribute to the probability that $H_n = t$. If the simple marginal probabilities of the missing parents are high, q_{np} will be low, and thus will help raise $P(H_n|S_1, \ldots, S_m, S_{np})$. If all the potential parents of H_n have previously been pruned, q_{np} will be 1 and thus have no effect on $P(H_n|S_1, \ldots, S_m, S_{np})$.

3.4 Implementation and Bayesian Inference

Our system updates the structure of the Bayesian network in response to each stroke the user draws. To perform this dynamic Bayesian network construction, we use an off-the-shelf, open source Bayesian network package for Java called BNJ [1]. Our system manages the hypotheses for the user's strokes as they are generated and pruned. When these hypotheses need to be evaluated (e.g., before they are pruned), our system creates a Bayesian network in BNJ by creating the necessary nodes and links as BNJ Java objects. Our system can then use a number of methods that are built-in to BNJ for reasoning about the probability of each node.

Generating and modifying the BNJ networks can be time consuming due to the exponential size of the conditional probability tables (CPTs) between the nodes. We use two techniques to improve the system's performance. First, BNJ networks are only generated when the system needs to evaluate the likelihood of a given hypothesis. This on-demand construction is more efficient than continuously updating the BNJ network because batch construction of the CPTs is often more efficient than incremental construction of these tables. Second, the system modifies only the portion of the BNJ network that has changed between strokes instead of creating it from scratch every time. The process of keeping track of the added and removed hypotheses adds a slight bookkeeping overhead, but this overhead is far less than the work required to regenerate the entire network after each stroke.

To determine the likelihood of each hypothesis, our system uses the BNJ network to find the marginal posterior probability for each node in the network. We experimented with several inference methods including the junction tree algorithm [52, 43], Gibbs sampling, and loopy belief propagation (loopy BP) [65, 79]. Both the junction tree algorithm and loopy BP produced meaningful marginal posterior probabilities, but after some experimentation we were unable to obtain useful results using Gibbs Sampling. We discovered that although the junction tree algorithm gave meaningful results, the networks produced by our system were often too complex for the algorithm to process in a reasonable amount of time. We found that when the junction tree algorithm produced a clique including more than 11 nodes the processing took too long to be acceptable. Unfortunately, for more complicated diagrams and domains, clique sizes greater than 11 were quite common.

Fortunately, we found loopy BP to be quite successful for our task. Although the algorithm is not guaranteed to converge to correct values, we found that on our data the algorithm almost always converged. There were probably only two or three instances in hundreds of tests where the values did not converge. We initialized the messages to 1 and ran the algorithm until node values were stable to within 0.001.

Loopy BP was significantly faster than the junction tree algorithm, but for complex data it was still occasionally slower then we wished. To speed up the system's performance, we added two restrictions. First, we terminated the belief propagation algorithm after 60 seconds of processing if it had not converged by this time. This restriction was needed only a about a dozen times in the 80 circuit diagrams we processed, but it prevented the rare case were belief propagation took 20 minutes to converge. Second, we allowed each node to have no more than 8 parents (i.e., only 8 higher-level hypotheses could be considered for a single hypothesis). This restriction ensured a limit on the complexity of the graphs produced by the system. For the family tree domain, this limitation had no effect on the system's performance because the system never generated more than 8 higher-level hypotheses for a lower-level hypothesis. However, in the circuit domain, higher-level hypotheses were occasionally prevented from being considered due to this limitation. For complex domains such as circuit diagrams, we will need to work on finding more efficient inference algorithms to allow the system to process more complex networks in a reasonable amount of time. We will also explore other methods of simplifying the network structure that do not prevent the system from considering possibly correct hypotheses.

3.5 Summary

In this chapter we have described a model for dynamically constructing Bayesian networks to represent varying hypotheses for the user's strokes. Our model, specifically developed for the task of recognition, allows both stroke data and contextual data to influence the probability of an interpretation for the user's strokes. Using noisy-OR, multiple potential higher-level interpretations mutually influence each other's probabilities within the dynamically constructed Bayesian network. Finally, we provide a method of incorporating the influence of nodes not present in the network to allow these networks to be simplified without affecting their semantics.

Although this model gives us a way to reason about possible hypotheses for the user's strokes, it does not specify how to generate these hypotheses. The process of hypothesis generation is the subject of the next chapter.

Chapter 4

Hypothesis Generation

In our system, recognition is a two-stage process involving hypothesis generation and hypothesis evaluation. The previous chapter described how hypotheses are evaluated using dynamically constructed Bayesian networks. The focus of this chapter is how these candidate hypotheses are generated.

The major difficulty in hypothesis generation is to generate the correct interpretation as a candidate hypothesis without generating too many hypotheses to consider in real time. A naive approach to hypothesis generation would be simply to attempt to match all shapes to all possible combinations of strokes, but this approach would produce an exponential number of interpretations. Our method of evaluating partial interpretations allows us to use a bottom-up/top-down generation strategy that greatly reduces the number of hypotheses considered but still generates the correct interpretation for most shapes in the sketch.

Our hypothesis generation algorithm has three basic steps, executed after each stroke the user draws.

1. **Bottom-up Step**: The system parses the newest stroke into one or more primitive objects using a domain-independent recognition toolkit developed in previous work [71]. Based on this low-level classification, the system hypothesizes higher-level interpretations for this stroke even even if not all the subshapes of the higher-level pattern have been found.

2. **Top-down Step**: The system attempts to find subshapes that are missing from the partial interpretations generated in the bottom-up step, often by reinterpreting strokes that are temporally and spatially proximal to the proposed shape.

3. **Pruning Step**: The system prunes off unlikely interpretations to keep the number of interpretations manageable.

Although our algorithm seems straightforward, there are a number of specific challenges that must be addressed to make it work. First, in the bottom-up step, the system must have an effective policy for instantiating hypotheses. One part of this hypothesis is specifying a threshold for generating a new hypothesis, e.g., requiring a minimum number of subshapes to be present and constraints to be met. Second, when generating and filling in hypotheses, the system must have a method for deciding how and where a shapes fits into a higher level shape. For example, an arrow is made of three lines. When hypothesizing that a particular line is part of an arrow, the system must also determine which part of the arrow that line best corresponds to. Because bindings can be ambiguous before all the subparts of a higher level interpretation are present, the system should be able to shuffle bindings in response to new information. The third challenge arises because the system uses high-level interpretations to recover from low-level interpretation errors, and, hence, it must have a specific method for reinterpreting strokes in response to incomplete higher-level interpretations. Fourth, the system relies on a variety of pruning methods to manage the number of hypotheses. In addition to pruning hypotheses with a low score in the Bayesian network, the system also prunes old, incomplete hypotheses and hypotheses containing redundant information. Finally, once the interpretations have been generated and scored, the system must choose a consistent set of interpretations for the user's strokes from among the many conflicting possible interpretations.

To further discuss each of the challenges above we again return to the example from Chapter 1 in which the user has begun to draw a family tree diagram (reproduced here as Figure 4-1).

4.1 Hypothesis Creation

When the user draws a new stroke, it is passed first to a domain-independent, low-level classification toolkit [71]. This toolkit classifies the stroke as one of the following: a line, an arc, a series of connected lines (a *polyline*), a series of Bezier curves, a combination of lines and curves, or none of the above. Unlike previous recognition toolkits, this recognition system is capable of breaking the stroke into a number of primitive shapes (e.g., a series of lines) rather than classifying it as a single shape. This ability affords the user more freedom in drawing her shapes, as she does not have to draw each primitive shape with a single stroke.

Based on low-level interpretations of a stroke, the bottom-up step generates a set of hypotheses

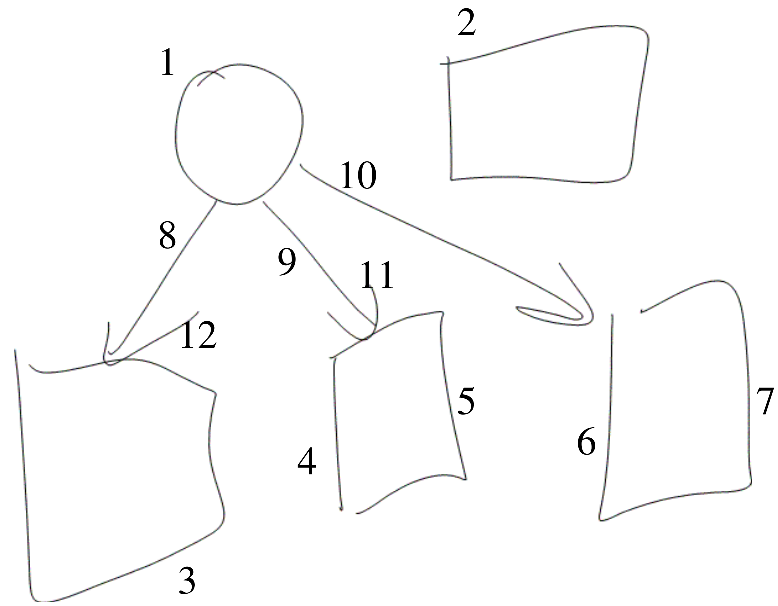


Figure 4-1: The partial sketch of a family tree from Chapter 1.

to be evaluated using the Bayesian network mechanism presented in the previous section. For example, in the example in Figure 4-1, the user's first stroke is correctly identified by the low-level recognizer as an ellipse, and from that ellipse the system generates the interpretation *female*.

The proposed interpretations are called *templates* that have a *slot* for each subshape. In this example, the ellipse hypothesis was filled into the ellipse slot in the female template, and there are no remaining empty slots. However, in other cases a template will only be partially filled, and future interpretations may be placed in empty slots. For example, when the user draws Stroke 6, the system generates a partially filled quadrilateral hypothesis, into which the two lines from Stroke 7 may be filled.

Naive bottom-up interpretation, in which the system generates every possible higher-level hypothesis for each low-level shape, can easily generate too many hypotheses to considered in real-time. In addition, recall that each higher-level interpretation for a given shape becomes a parent of that node in the Bayesian network, leading to a large and highly connected graph structure, making inference far more time consuming. The system relies on a number of strategies to limit the number of generated partial interpretations.

The first strategy is that the system does not generate a compound shape that consists of more than one subshape until at least two subshapes and one constraint between those subshapes are present. When the system attempts to fit two or more shapes into a single higher-level shape, it first checks to ensure that a minimum number of constraints between the subshapes appear to be met. Our model allows the system to use context to recognize shapes despite the fact that that some of the constraints required for that shape may not appear to hold in isolation (e.g., the constraint that the lines formed by Strokes 6 and 7 are coincident in the top left corner of the quadrilateral in Figure 4-1). To enable this context-based evaluation, the system cannot rule out interpretations simply because some of the constraints appear to fail in isolation. On the other hand, the system should not generate a hypothesis if too many constraints are violated.

To determine when to generate a hypothesis, the system needs a way to measure how well each constraint holds based on the stroke data. For each constraint, we define a corresponding property that can be measured objectively from the stroke data, as discussed in 2.2.1. For example, the property related to the constraint "parallel" is the angle between the two lines. Each property has a continuous value and a different range. For example, the angle between two lines is a real number between 0 and 180, while the ratio between the lengths of two lines is a positive real number. To allow the system to compare values, we translate the value of each property into a standardized measurement of how well a constraint holds, which we call the *constraint measurement*.

In determining the number of values for the constraint measurement, our goal was to choose the minimum number that would carry sufficient information for hypothesis generation. As mentioned previously, two values is insufficient because it is often impossible to tell whether or not a constraint holds based on low-level data alone. On the other hand, it is unlikely that there are ten meaningful categories for how well a constraint holds based on the stroke data. We collected data for each constraint by asking 27 subjects to draw simple shapes and constraints. For each constraint, we measured the value of its corresponding property for sketches where the property was present and sketches were the property was not present. Based on this data, we discovered three meaningful values for each constraint measurement. The first values corresponds to the region in which the value of the property strongly supports the constraint. For property values in this region the constraint is almost always true. The second value corresponds to the region in which the property only weakly supports the constraint. In this region the constraint is sometimes true, but is sometimes false. The third value corresponds to the region in which the property contradicts the constraint. In this region, the constraint is rarely, if ever, true. We assign numerical values to each region: zero (property strongly supports constraint), one (property weakly supports constraint), and two (property contradicts constraint).[1] Appendix B gives the exact properties and thresholds we use in determining how

[1]Our representation allows future constraint measurements to have more than three values and be comparable to our three-valued constraint measurements. Zero always represents the region in which the property most strongly supports

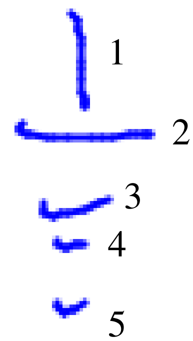


Figure 4-2: A sketched ground symbol.

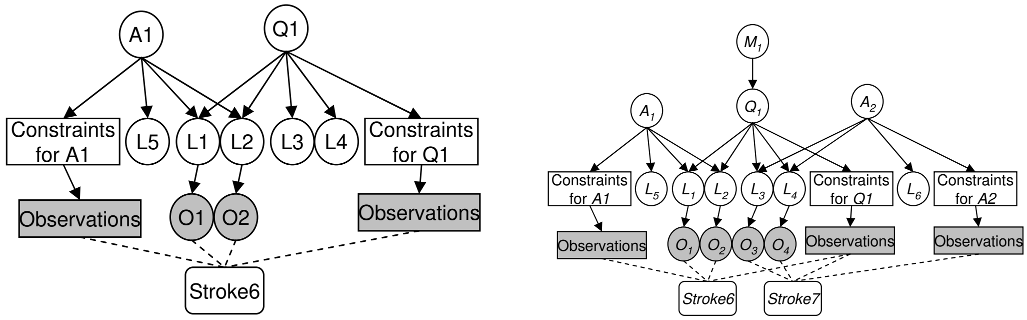
well each constraint holds.

For the system to generate a single template containing two or more objects, at least half of the necessary constraint measurements between those objects must be zero, and no more than one may be two. This threshold was determined empirically. For example, in Figure 4-2, the system would generate a ground template involving Strokes 1 and 2 even though they do not touch, because the stroke data strongly supports the constraint that they are perpendicular. However, the system would not generate a battery template for Strokes 1 and 4 because two of the necessary constraints (that the lines are parallel and that they are next to one another) are contradicted by the stroke data.

The second strategy the system uses to control the number of generated hypotheses is that the system does not generate higher-level interpretations for interpretations that are only partially filled. For example, the two lines generated from Stroke 6 in Figure 4-1 results in two partial hypotheses: an arrow and a quadrilateral (Figure 4-3(a)). The system does not generate any higher-level hypotheses for either the arrow or the quadrilateral because neither is full. When the user draws the other two sides of the quadrilateral with her next stroke (Stroke 7), the system fills in the rest of the quadrilateral hypothesis and generates the next higher-level hypothesis for the now complete quadrilateral, i.e., a male (Figure 4-3(b)). (Notice, however, that this male hypothesis will not be combined yet with the female hypothesis generated from Stroke 1 to form a mother-son domain pattern because there are no constraints directly constraining the female relative to the male in the mother-son template, as per the first strategy, described above.)

Third, each time a new shape is hypothesized, the system first attempts to fit this shape into existing templates that use that shape before creating new templates. In Figure 4-3, the system fit

the constraint.



(a) Hypothesis generated from Stroke 6.

(b) Hypotheses generated from Strokes 6 and 7.

Figure 4-3: Hypotheses generated from Strokes 6 and 7 in the sketch in Figure 4-1

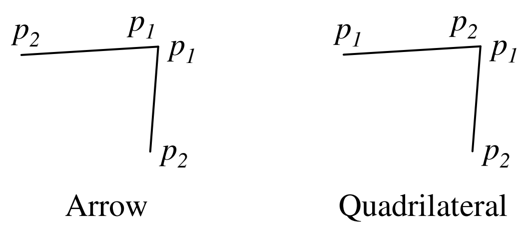


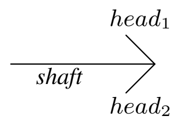
Figure 4-4: Two labellings of p_1 and p_2 that allow the same two lines to be recognized as part of different higher-level shapes.

L_3 and L_4 into the template for quadrilateral Q_1 and did not create a new quadrilateral template. However, it was unable to fit L_3 and L_4 into A_1, and created a second arrow template (A_2) to account for lines L_3 and L_4.

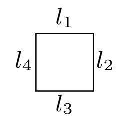
When an interpretation can be fit into more than one slot in a higher-level template (e.g., lines L_1 and L_2 might be the head of an A_1, or one side of the head and the shaft), the system arbitrarily chooses one valid fit rather than generating one hypothesis for each potential fit. We discuss below how the system can shuffle shapes in a template when it attempts to fit more subshapes.

4.1.1 Subcomponent Labeling

Some constraints, such as the coincident constraint, constrain the subcomponents of a pair of shapes relative to one another. This subcomponent reference presents a challenge for the system because there is often more than one way to label the subcomponents in a shape. For example, consider the two lines in Figure 4-4. They might be the head of an arrow, or the side of a quadrilateral. However, given the descriptions for an arrow and for a quadrilateral in Figure 4.1.1, the lines must have their endpoints labeled differently to be fit into the two templates.



DEFINE ARROW
(**Subshapes**
  Line *shaft*
  Line $head_1$
  Line $head_2$)
(**Constraints**
  (coincident *shaft*.p_1 $head_1.p_1$)
  (coincident *shaft*.p_1 $head_2.p_2$)
  (equalLength $head_1$ $head_2$)
  (larger *shaft* $head_1$)
  (acuteAngle $head_1$ *shaft*)
  (acuteAngle $head_2$ *shaft*))



DEFINE QUADRILATERAL
(**Subshapes**
  (vector Line $l[4]$))
(**Constraints**
  (coincident $l[i].p_2$ $l[i+1].p_1$)
  (coincident $l[4].p_2$ $l[1].p_1$))

Figure 4-5: Descriptions for an arrow and a quadrilateral.

Subcomponent labeling is a challenging problem that our system cannot yet handle in a general case, as we discuss in the next chapter. However, this problem is most prominent in our domains for the endpoints of lines. For the specific case of line endpoint labeling, we have solved this problem by creating two hypotheses for every line that is not part of a polyline—one with each of the two possible end point labellings.

4.1.2 Polylines

When the system processes polylines, it assumes that all the lines in a single polyline will be used in one interpretation. For example, in Figure 4-3 the system did not generate marriage link templates for any of lines L_1–L_4 (resulting from Strokes 6 and 7) because they were all part of polyline objects. While this assumption does not always hold, in practice we find that it is often true and greatly reduces the number of possible interpretations.

Furthermore, in matching polylines into higher-level compound shapes, the system takes advantage of the connectivity of the lines in a polyline to narrow down the possible bindings from lines to slots. Because of the endpoint labeling problem discussed above, the number of theoretically possible sets of lines in a polyline with n lines is 2^n. However, by matching the connection points in the polyline with the connection points in the template, the system can consider a greatly reduced

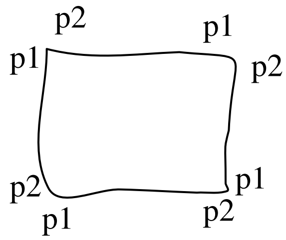


Figure 4-6: A labeling of $p1$ and $p2$ for each line in a polyline that allows the system to recognize a quadrilateral.

set of these bindings. For example when fitting a polyline into a quadrilateral template, the system need only try endpoint labellings that put all the lines in the same direction, as shown in Figure 4-6, and need not try any other endpoint labellings.

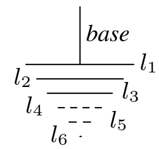
4.2 Slot Binding

The second challenge in hypothesis generation is determining the correct mapping between lower level shapes and the subparts in higher-level shapes. In this section, we consider the challenges of this task and present two algorithms for performing this mapping. We compare the relative performance of the two algorithms in the next chapter.

In some cases, particularly when the user draws the subparts of a symbol in a predicted drawing order, filling shapes into a template is straightforward. For example, the description of a ground symbol is shown in Figure 4-7. As the user draws the ground symbol in Figure 4-8(a), the system fits the first line into the *base* slot in the template, the next line into the l_1 slot, the next line into the l_2 slot, and so on.

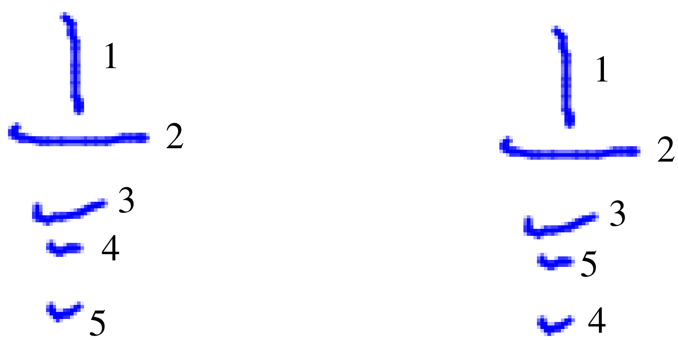
On the other hand, the stroke ordering in Figure 4-8(b) results in an incorrect, but seemingly reasonable, binding between lines and slots in which the line recognized from Stroke 4 is filled into slot l_3. Consequently, when the user draws Stroke 5, it cannot be correctly bound to slot l_3. In order to recognize the ground symbol, the system must shuffle the components that have already been bound to this template.

Shuffling the subparts within a template can be time consuming. With a straightforward approach, the process of finding the best fit for m strokes into s slots is $\binom{m}{s}$. At first, it would seem that stroke orderings that lead the system to produce incorrect mappings between subshapes and template would be relatively rare, so the system should not have to shuffle the subparts in a template very often. Unfortunately, there is a more severe consequence of the above situation. In the course



DEFINE GROUND
(**Subshapes**
  Line *base*
  (vector Line $l[3, 6]$)
(**Constraints**
  (perpendicular *base* l_1)
  (touches *base* l_1 *base*.p_2)
  (parallel $l[i]$ $l[i + 1]$)
  (smaller $l[i + 1]$ $l[i]$)
  (nextTo $l[i]$ $l[i + 1]$)
  (aligned $l[i]$ $l[i + 1]$))

Figure 4-7: The ground symbol from the Circuit Diagram domain.



(a) An expected stroke order. (b) An unexpected stroke order.

Figure 4-8: Two different stroke orders subjects used when drawing a ground symbol.

of generating and filling templates, the system often tries to fit a shape into a template into which it does not fit at all. The system cannot immediately distinguish between the cases where an object will fit into a template if the subparts already bound to that template are shuffled and the cases where an object cannot be fit into a template. In order to determine the difference, the system must attempt to shuffle the components in a template every time an object does not fit into that template, to ensure that it really does not fit.

The system must do this shuffling because constraints between subcomponents are calculated on demand, as the system attempts to fit a new shape into a template. Thus, the system cannot tell if

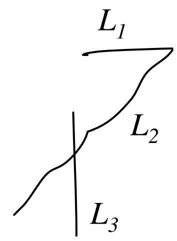


Figure 4-9: Two lines that form part of an arrow (L_1 and L_2) and a third line (L_3) that is obviously not part of the arrow.

a shape meets the necessary constraints to be fit into *any* slot in the template without attempting to fit it into each slot. In most of these cases, however, considering the constraints before fitting the the shape into the template would allow the system to rule out immediately a number of potential ways to bind the shape into template. For example, the line labeled L_3 in Figure 4-9 could not possibly be part of any arrow interpretation involving L_1 and L_2 because it has no point which is coincident with either line.

While relying on constraints can help the system prune the search space for hypotheses, there are two challenges to using this approach. First, relying on constraints may cause the system to miss a correct interpretation because, due to signal-level noise, some constraints do not appear to hold based on stroke data alone. Second, pruning the search space using constraints needs to take less time than trying the different mappings between slots and shapes for it to be an effective strategy. Meeting this requirement is not always trivial. The constraints in a template refer to slots, which are essentially variables. Given a set of constraints that hold between shapes in the drawing, finding a mapping between the shapes in those constraints and the variables in the template constraints is an NP-hard problem.

Taking these challenges into consideration, we use the following method to prune the search space based on constraints when shuffling the components in a template. Given a new shape, S_n, the system gathers all of the existing shapes within a spatial and temporal bound that do not rely on the same strokes as the new shape.[2] For each object in this set, S_e, the system calculates the values of each applicable constraint between S_e and S_n. If the value of this constraint is 0, the system records that the constraint, C_i, holds between S_e and S_n. For those constraints that are commutative (e.g., parallel), the system calculates the value of the constraint in both directions. At the end of this process the system has accumulated only the constraints that are strongly supported by the stroke

[2]Two stroke shapes cannot be fit into a higher-level interpretation if they are derived from the same stroke because these shapes represent competing interpretations for the user's stroke.

Template Constraints	Object Constraints
C_1: (coincident *shaft*.p_1 $head_1.p_1$)	(shorter L_1 L_2)
C_2: (coincident *shaft*.p_1 $head_2.p_2$)	(coincident $L_1.p_1$ $L_2.p_1$)
C_3: (equalLength $head_1$ $head_2$)	(acute-angle $L_1 L_2$)
C_4: (shorter $head_1$ *shaft*)	(shorter L_3 L_2)
C_5: (acuteAngle $head_1$ *shaft*)	(shorter L_1 L_3)
C_6: (acuteAngle $head_2$ *shaft*)	

(a) Constraints

	$head_1$				$head_2$			*shaft*				
	C_1	C_3	C_4	C_5	C_2	C_3	C_6	C_1	C_2	C_4	C_5	C_6
L_1	x		x	x	x		x	x			x	
L_2	x		x		x	x		x		x	x	
L_3			x							x		

(b) Constraint Table

Table 4.1: The constraint table built while trying to fit the three lines in Figure 4-9 into an arrow template. Template constraints and detected object constraints are shown in the top table.

data.

To fit S_n into a template, t, that already contains shapes $S_1...S_k$, the system uses a voting algorithm inspired by template matching algorithms in computer vision [24]. The system constructs a table containing columns for each slot in t and rows for each shape $(S_n, S_1, \ldots S_k)$. Each column has a subcolumn for each constraint that pertains to that slot. Table 4.1(a) shows an example for the arrow template, given the sketch in Figure 4-9. Next, the system gathers all calculated constraints pertaining to $(S_n, S_1, \ldots S_i)$ (shown in Figure 4.1(a)). We distinguish between *object constraints*, which are the constraints determined to hold between existing shapes and *template constraints*, which are the constraints required by a template. For each object constraint, the system puts an "x" in the corresponding row for each column that pertains to the matching template constraint. For example, the first object constraint in the list in Table 4.1(a) results in two "x"s in the table: one in row L_1, column $head_1$-C_4, and the other in row L_2, column *shaft*-C_4. The full table in Table 4.2 shows the results of processing all the object constraints in Table 4.1(a). Finally, the system considers each row of the complete table and allows a component to be fit into any slot where it has satisfied at least 75% of the constraints. Notice that in this example, there is no such slot for either L_3 or L_2, so the system does not try to shuffle the components in the arrow template even though L_1 satisfies enough constraints to be fit into the slot for $head_1$

Intuitively, we expect that the above algorithm will improve processing time when there are relatively few constraints that are satisfied between shapes in the sketch. In practice we find that for simple sketches this is indeed the case. For more complicated sketches, as we will see in the next chapter, running time is dominated by the Bayesian network inference. We refer to the above algorithm as the constraint-based template generation algorithm, and the original algorithm as the slot-based template generation algorithm. We compare the performance of these two algorithms empirically in the next chapter.

4.3 Recovering from Low-level Errors

In the bottom-up template generation steps, the system considers only the best low-level interpretation generated by the low-level toolkit. Often on messy data, it is difficult or impossible to correctly identify low-level shapes without considering the stroke in the context of the larger sketch, and the low-level toolkit often generates an incorrect low-level interpretation.

The top-down step allows our system to recover from low-level recognition errors by considering two common sources of error made by the low-level toolkit. First, the system attempts to reclassify a stroke as a line, arc or an ellipse to fill in nearby partially filled hypotheses requiring these shapes. Second, the system attempts to re-segment polylines to fit them into higher-level shapes. For example, in the drawing in Figure 4-1, Stroke 3 is incorrectly, but reasonably, parsed into 5 lines by the low-level recognizer. Because the system does not know about 5-line objects, but does know about things that contain fewer than 5 lines, it attempts to re-segment the stroke into 2 lines, 3 lines and 4 lines (with a threshold on acceptable error). It succeeds in re-segmenting the stroke into 4 lines and successfully recognizes the lines as a quadrilateral. The portion of the interpretation network involved with parsing this stroke is given in Figure 4-10. Although the 4-line fit is not perfect, the network allows the context of the quadrilateral in addition to the stroke data to influence the system's belief in the 4-line interpretation. Also note that the 5 lines from the original segmentation are still present in the interpretation network.

The top-down step also fills in hypotheses generated by the bottom-up step in order to allow the bottom-up hypothesis to use a simpler and faster method for generating hypotheses. Recall that a higher-level hypothesis is not proposed until there are at least two subshapes with a constraint between them present in the sketch. Consequently, in Figure 4-1, the mother-son relationship between the female and the rightmost male is not proposed until after the arrow between the two shapes is drawn (Stroke 10). The system proposes an arrow, and then a child-link, interpretation for Stroke

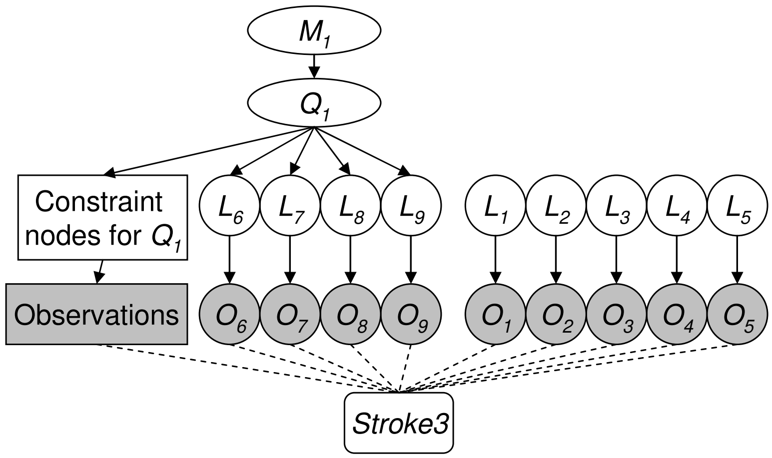


Figure 4-10: The part of the interpretation network for the interpretation of the drawing in Figure 4-1 that pertains to the two line segmentations for Stroke 3.

10 and then looks for other shapes with which to combine the child-link to produce higher-level domain patterns. To save time, it tries to combine the child-link interpretation with only one other interpretation (e.g., either the female or the male, but not both), rather than considering all possible sets of shapes that might be combined to form higher-level interpretations. It then instantiates a mother-son domain pattern that contains only the female and the child-link, but not the male. The top-down step can then fill in the male hypotheses to complete the domain pattern correctly.

In the final form of low-level error recovery, the top-down step collapses overlapping polylines in order to recognize shapes such as the arrow created by Stroke 10 in Figure 4-1. For any lines in a polyline with an angle less than 15 degrees, the system proposes a single line to account for both lines. As long as there is a higher level template into which this collapsed polyline may be fit, the system keeps this collapsed version of the polyline. If the collapsed polyline cannot be fit into a higher level template, the system prunes the collapsed hypothesis.

4.4 Pruning Hypotheses

The system controls the number of interpretations in the network through pruning. The system prunes hypotheses based on three criteria: likelihood, age and redundancy. The system prunes any hypotheses that are not likely interpretations for the user's strokes. The pruning step compares all competing interpretations for a given node and prunes any that are more than a set threshold below the best interpretation. Next, the system prunes incomplete hypotheses that are older than a given threshold under the assumption that if the user has not completed a shape within a set period of time, it is unlikely that this shape is the correct interpretation for the user's strokes. The system prunes any incomplete hypotheses that have not had a slot filled with shapes resulting from the user's previous two strokes, and it prunes all domain patterns that have not been modified in the user's previous four strokes. Note that the top-down step is capable of regenerating these pruned hypotheses if the user returns to complete the shape. Finally, because each line is represented in the system twice (each with a different labeling for endpoints p_1 and p_2), the system often contains a number of redundant interpretations. The system prunes any interpretation that is more than a set threshold below another incomplete interpretation of the same type that uses the same subshapes.

4.5 Selecting an Interpretation

As each stroke is drawn the sketch system uses a greedy algorithm to select the best interpretation for the sketch. It queries the Bayesian network for the strongest complete domain shape interpretation, sets aside all the interpretations inconsistent with this choice, chooses the next most likely remaining domain interpretation, and so forth. It leaves uninterpreted strokes that are part of partial hypotheses.

Although the system selects the most likely interpretation at every stroke, it does not eliminate other interpretations. Partial interpretations remain and can be completed with the user's subsequent strokes. Additionally, the system can change its interpretation of a stroke when more context is added. While interpretations are selected after every stroke, determining when to display these interpretations to the user is by itself an interesting and difficult question of human computer interaction. We further explore this issue in Chapter 6.

4.6 Summary

This chapter presented our approach to hypothesis generation for sketch recognition. We introduced several techniques for controlling the number of hypotheses generated. We introduced two techniques for generating hypotheses: the slot-based approach and the constraint based approach. Finally, we showed how to use partial hypotheses to reinterpret strokes that were incorrectly recognized by the low-level classifier. In the next chapter we illustrate the strengths and weaknesses of each of these ideas by exploring the system's performance on real-world data.

Chapter 5

Evaluation

Motivated by the desire to construct a multi-domain sketch recognition engine for use in early-stage design tools, we have presented a new domain-variable context-based approach to sketch recognition. A system that is intended to be broadly useful for the construction of early-stage design tools should provide:

- **Recognition accuracy**: The system should be able to accurately recognize freely drawn, messy sketches. It should not necessarily require feedback from the user to guide interpretation; providing recognition feedback can be distracting.
- **Real-time performance**: The system should recognize the user's sketches in real time so the designer can interact with recognized diagrams at any point during the design process.
- **Domain flexibility**: The system should be applicable to a variety of domains without requiring programming or extensive amounts of training data.

Although we have not yet succeeded in building a system that meets each of these requirements completely, we have taken a number of substantial steps toward each goal and, in doing so, exposed several remaining challenges. We show how our system is applied to two non-trivial domains—family trees and circuits—and discuss the inherent challenges in each domain. We provide examples that show our system is capable of using context to resolve ambiguity in messy sketches, we present quantitative recognition and running time performance results on complex, messy sketches from each domain, showing that the use of context improves recognition performance. Finally, an examination of the system's recognition errors and running time bottlenecks illuminates the remaining challenges in building a multi-domain sketch recognition engine.

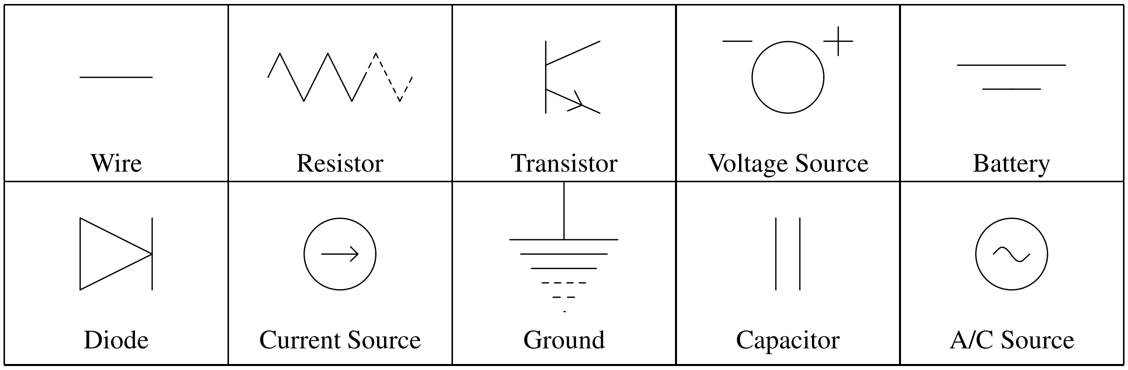


Table 5.1: The symbols in the circuit domain.

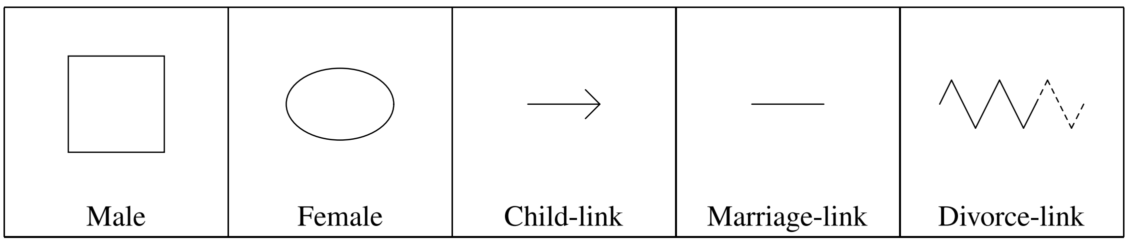


Table 5.2: The symbols in the family-tree domain.

5.1 Applying SketchREAD

While there exist a number of domain-specific recognition systems (e.g., [4, 36, 49]), there have been relatively few attempts to build a multi-domain system. Weisman [78] and Lank [50] proposed domain-retargetable systems, but their systems require a significant amount to programming to apply them to a domain. The contribution of our work, and complementary work by Hammond and Davis [38], is that our engine can be applied to a particular domain without any programming.

To show that SketchREAD achieves this goal, we applied it to two non-trivial domains—family trees and circuits—and found that it is capable of recognizing sketches in both domains without reprogramming. Applying SketchREAD to a particular domain involves two steps: specifying the structural descriptions for the shapes in the domain and specifying the prior probabilities for the domain patterns and any top-level shapes not used in domain patterns, i.e., those that will not have parents in the generated Bayesian network. (Refer to Chapter 3 for details on how probabilities are assigned to other shapes.) For each domain, we wrote a description for each shape and pattern in that domain. Tables 5.1 and 5.2 give the shapes in each domain, and Appendix A gives full descriptions for all the shapes and the domain patterns.

We specified only shape descriptions (no domain pattern descriptions) for the circuit diagrams because of a limitation in how our implemented system handles subcomponent labeling (discussed in Section 5.6.1). We describe below how the lack of domain patterns for circuit diagrams may have affected the system's performance. Note, however, that the domain shapes themselves still provide a context in which to interpret lower-level interpretations and constraints in the circuit domain.

We hand-estimated priors for each domain pattern and top level shape, based on our intuition about the relative prevalence of each shape. For example, in our family-tree diagrams, we estimated that Marriage relationships were much more likely than Partnership relationships, and we set $P[Mar] = 0.1$ and $P[Part] = 0.001$. These priors represent the probability that a group of strokes chosen at random from the page will represent the given shape and, in most cases, should be relatively low. The priors we used in our tests are given in Appendix B, but as discussed below, the system's recognition performance was relatively insensitive to the exact values of these priors.

Having to set priors by hand could be difficult for a system designer who may know very little about probabilities. We suggest two ways to assist the system designer with this task. First, through experimentation, we found that recognition performance to be relatively insensitive to the exact values of these priors. For example, in the circuit diagrams, increasing all the priors by an order of magnitude did not affect recognition performance. Instead what matters is the relative values of the prior probabilities. Based on this observation, we could build a system that translates the system designer's knowledge about the relative strengths of each domain pattern and top-level shape into the prior probabilities used by the system. As a second approach, the prior probabilities could be learned from data. While SketchREAD does not require labeled training data, if the system designer has access to such data, SketchREAD should use the data effectively.

5.2 Data Collection

SketchREAD is capable of recognizing simple sketches nearly perfectly in both the family-tree and circuit domains, but we wanted to test its performance on more complex, real-world data. There is no standard test corpus for sketch recognition, so we collected our own sketches and have made them available online at `http://rationale.csail.mit.edu/ETCHASketches` to encourage others to compare their results with those presented here.

Our goal was to collect sketches that were as natural and unconstrained as the types of sketches people produce on paper to test the limits of our system's recognition performance. To collect these sketches, we used a data collection program for the Tablet PC developed by others in our group that allows the user to sketch freely and displays the user's strokes exactly as she draws them, without performing any type of recognition. Most of our users had played with a Tablet PC before they performed our data collection task but had never used one for an extended period of time. None used any type of pen-based computer interface on a regular basis. The users first performed a few warm-up tasks, at the end of which all users expressed comfort drawing on the Tablet PC.

To collect the family-tree sketches, we asked each of 10 users to draw her family tree using the symbols presented in Table 5.2. Users were told to draw as much or as little of the tree as they wanted and that they could draw the shapes however it felt natural to them. Because erasing strokes introduces subtleties into the recognition process that our system is not yet designed to deal with, users were told that they could not erase, and that the exact accuracy of their family-tree diagram was not critical.

We then recruited 10 different subjects with basic knowledge of circuit diagram construction and showed them examples of the types of circuits we were looking for. After a warm-up task, subjects were instructed to draw several circuits. We specified the number and types of components to be included in the circuit and then asked them to design any circuit using those components. Subjects were instructed not to worry about the functionality of their circuits, but that they should try to produce realistic circuits. We collected 80 diagrams in all.

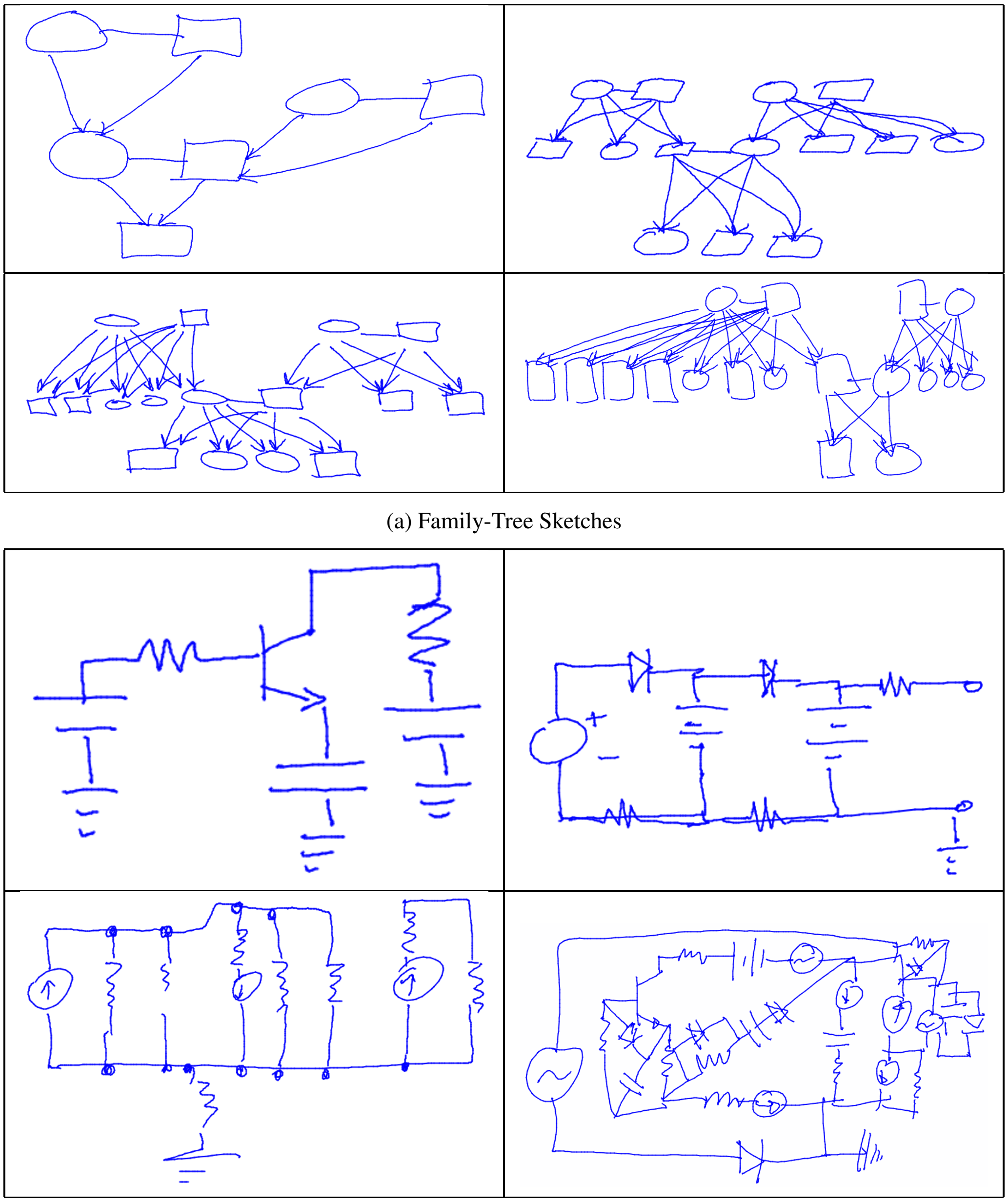
Figure 5-1 shows the range of complexity of the sketches in each domain. In the family-tree domain, we collected ten sketches of varying complexity containing between 24 and 102 strokes (mean=50, s.d.=26) and between 16 and 60 symbols (mean=34, s.d.=15). In the circuit domain, we collected 80 diagrams with between 23 and 110 strokes (mean=39, s.d.=13) and between 10 and 63 symbols[1] (mean=24, s.d.=8).

One limiting assumption that SketchREAD currently makes is that the user will not draw more than one symbol with a single stroke. Unfortunately, in drawing circuit diagrams, users often draw many symbols with a single stroke, e.g., a series of resistors and wires. To allow SketchREAD to handle the circuit diagrams, we manually divided strokes containing multiple objects. This is clearly a limitation of our current system; we discuss below how it might be handled.

5.3 Evaluation Methodology

In evaluating the performance of our system, direct comparisons with previous work are difficult, as there are few (if any) published results for this type of recognition task. The closest published sketch

[1]Wires drawn with a single stroke were counted as only one symbol, even if they were bent.



(a) Family-Tree Sketches

(b) Circuit Sketches

Figure 5-1: Examples that illustrate the range of complexity of the sketches collected.

recognition results are for the Quickset system, which also uses top-down information (through multi-modal input) to recognize sketched symbols, but this system assumes users will draw all of a single shape and then pause, making its recognition task different from ours [83].

We compared SketchREAD's recognition performance with the performance of a strictly bottom-up approach of the sort used in previous systems [4, 63]. This strictly bottom-up approach combined low-level shapes into higher-level patterns without top-down reinterpretation. Even though our baseline system did not reinterpret low-level interpretations, it was not trivial. It used the Bayesian

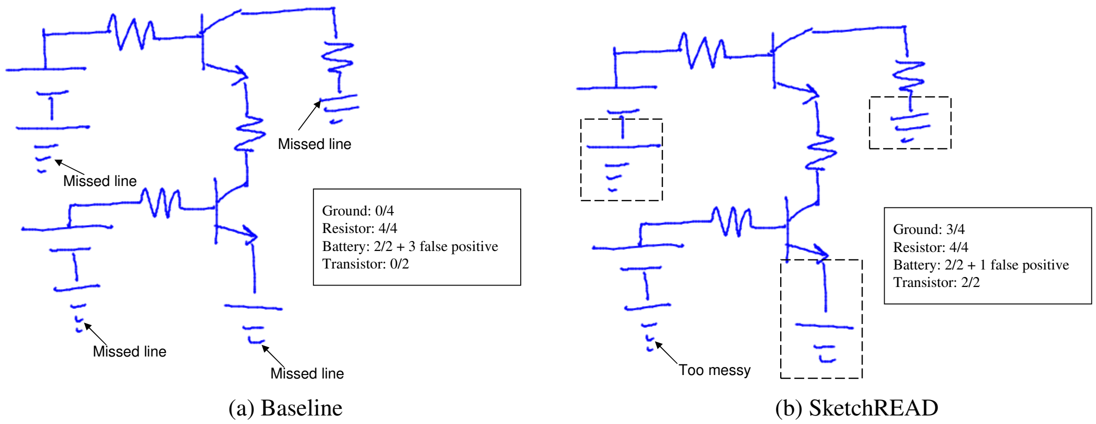


Figure 5-2: Recognition performance example. Numbers in the boxes give overall performance (number correct/total).

network mechanism to resolve inherent ambiguities in the drawings (e.g., are two strokes a battery or part of a ground symbol?). Also, because the bottom-up step generates hypotheses even when not all required constraints are met, the baseline system could cope with some noise in the sketch.

SketchREAD is designed to alleviate the need for domain-specific recognition systems. Although we do not present a direct comparison between SketchREAD and a domain-specific recognition system, we discuss SketchREAD's strengths and limitations relative to such a system in Section 5.6.3.

5.4 Qualitative Performance

Qualitative analysis of the system's performance reveals that our system's hypothesis generation and hypothesis evaluation mechanisms behave as expected. The system can successfully reinterpret low-level strokes to generate correct hypotheses that were missed in the bottom-up step. In addition, the Bayesian network mechanism successfully aggregates information from stroke data and context, resolves inherent ambiguities in the drawing, and updates its weighting of the various existing hypotheses as new strokes are drawn.

Figure 5-2 illustrates how our system is capable of handling noise in the sketch and recovering from missed low-level interpretations. In the baseline case (Figure 5-2(a)), one line from each ground symbol was incorrectly interpreted at the low-level, so the correct ground symbol hypotheses were never generated. SketchREAD was able to reinterpret those lines using the context of the ground symbol in three of the four cases to identify the symbol correctly (Figure 5-2(b)). In the

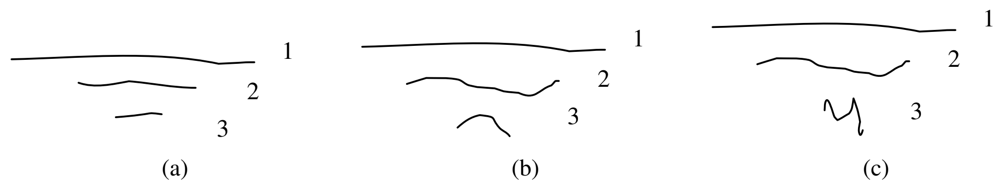


Figure 5-3: Part of three possible ground symbols.

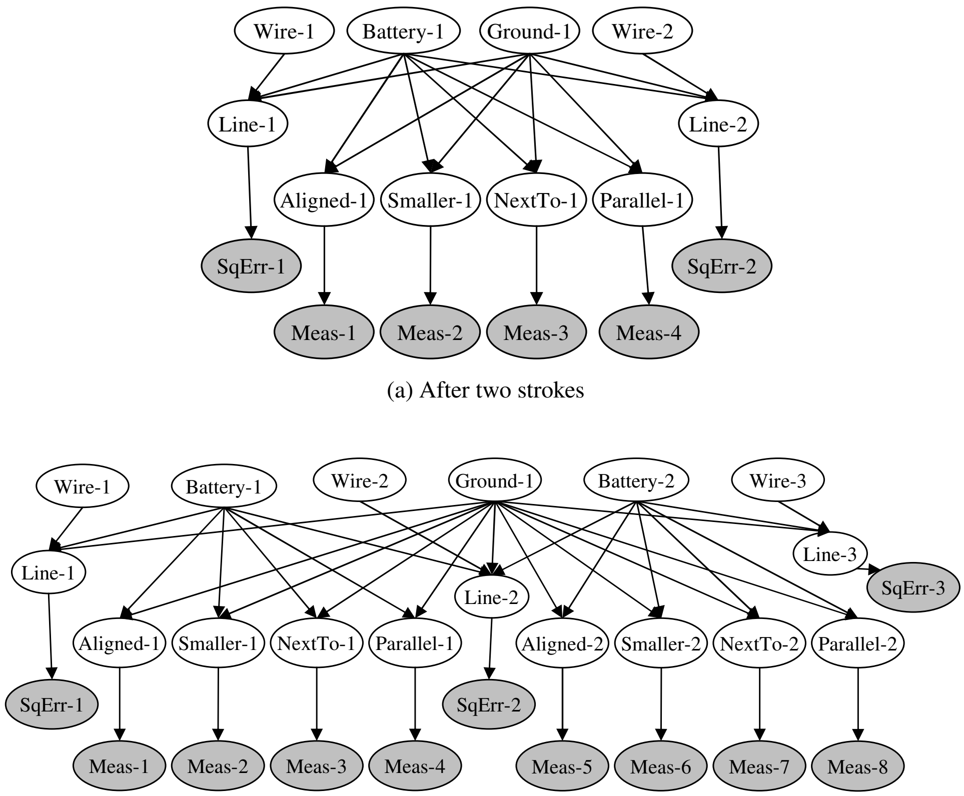
fourth case, one of the lines was simply too messy, and SketchREAD preferred (incorrectly) to recognize the top two lines of the ground symbol as a battery.

The system's performance depends on both its ability to generate the correct hypotheses and its ability to weight the correct hypotheses higher than the incorrect hypotheses. We show through an example that the Bayesian network scores hypotheses correctly in several respects: it prefers to group subshapes into the fewest number of interpretations, it allows competing interpretations to influence one another, it updates interpretation strengths in response to new stroke data, and it allows both stroke data and context to influence interpretation strength.

To illustrate the points above, we consider in detail how the system responds as the user draws the three sets of strokes in Figure 5-3. To simplify the example, we consider a reduced circuit domain in which users only draw wires, resistors, ground symbols and batteries. Figures 5-4(a) and 5-4(b) show the Bayesian networks produced in response to the user's first two and first three strokes, respectively. Note that the structure of the network is the same despite the noise in some of the drawings[2]; the values of the shaded nodes change in response to the noise. Observed nodes may have one of three values: 0 (stroke data strongly supports the shape or constraint), 1 (stroke data weakly supports the shape or constraint), and 2 (stroke data does not support the shape or constraint).

For the clean ground symbol (Figure 5-3(a)), posterior probabilities are given in Table 5.3. In this case the stroke data strongly supports all the shapes and constraints, so all shaded nodes have value 0. After the first two strokes, the battery symbol and the ground symbol have equal weight. The fact that the ground and battery symbols have the same weight at this point might seem counterintuitive. After all, after two strokes the user has drawn what appears to be a complete battery, but has only drawn a small piece of the ground symbol. Recall, however, that the Bayesian network models not what has been drawn, but what the user *intends to draw*. After just two strokes, it is

[2]In Figures 5-3(b) and 5-3(c), even though the user's second and third strokes do not necessarily look like lines, the system will generate line a hypothesis for each of these strokes in the top-down step.



(a) After two strokes

(b) After three strokes

Figure 5-4: The Bayesian network produced after the first two and three strokes of the ground symbol in Figure 5-3.

just as likely that the user is in the process of drawing a ground symbol. After the third stroke, the ground symbol's probability increases because there is more evidence to support it, while the battery's probability decreases. The fact that the ground symbol is preferred over the battery illustrates that the system prefers interpretations that result in a fewer number of symbols (Okham's razor). Interpretations that involve more of the user's strokes have more support and get a higher score in the Bayesian network. The fact that the battery symbol gets weaker as the ground symbol gets stronger results from the "explaining away" behavior in this Bayesian network configuration: each of the low-level components (the lines and the constraints) are effectively "explained" by the existence of the ground symbol, so the battery is no longer needed to explain their presence.

Next, we show that small amounts of noise in the data are counteracted by the context provided by higher-level shapes. The slightly noisy second and third strokes (Figure 5-3(b)) cause the values of SqErr-2 and SqErr-3 to be 1 instead of 0 (all other shaded node values remain 0) and result in the node values given in Table 5.4(a). The probability of the ground symbol interpretation is still

Name	$P[Shape \| stroke\ data]$	
	after 2 strokes	**after 3 strokes**
Wire-1	0.4	0.4
Wire-2	0.4	0.4
Wire-3	N/A	0.4
Battery-1	0.51	0.09
Battery-2	N/A	0.09
Ground-1	0.51	0.95
Line-1	1.0	1.0
Line-2	1.0	1.0
Line-3	N/A	1.0
Aligned-1	1.0	1.0
Smaller-1	1.0	1.0
NextTo-1	1.0	1.0
Parallel-1	1.0	1.0
Aligned-2	N/A	1.0
Smaller-2	N/A	1.0
NextTo-2	N/A	1.0
Parallel-2	N/A	1.0

Table 5.3: Posterior probabilities for the network in Figure 5.4 for the sketch in Figure 5-3(a).

high, due to the fact that Line-1 and all of the constraints are still strongly supported by the data. In addition, the probabilities for the line nodes are high, indicating that the context provided by the ground symbol provides support for the line interpretations even when the evidence from the data is not as strong.

On the other hand, if the data is too messy, it causes the the probability of the higher-level interpretations to decrease. The sketch in Figure 5-3(c) causes the value of SqErr-2 to be 1, and SqErr-3 to be 2, and results in the posterior probabilities given in Table 5.4(b). In this case, Lines-1 and Line-2 are accounted for by Battery-1. The probability of Ground-1 decreases because the hypothesis Line-3 is contradicted by the user's third stroke.

5.5 Quantitative Results

In this section we present quantitative recognition and running time results for the family tree and circuit domains. Although SketchREAD does not perform perfectly on every sketch, its generality and performance on complex sketches illustrates its promise over previous approaches. In addition to comparing SketchREAD's performance to the performance of the baseline system, we also

Name	$P[Shape\|stroke\ data]$ after 2 strokes	$P[Shape\|stroke\ data]$ after 3 strokes
Wire-1	0.41	0.4
Wire-2	0.38	0.4
Wire-3	N/A	0.4
Battery-1	0.51	0.1
Battery-2	N/A	0.1
Ground-1	0.51	0.94
Line-1	1.0	1.0
Line-2	0.96	1.0
Line-3	N/A	1.0
Aligned-1	0.98	1.0
Smaller-1	0.98	1.0
NextTo-1	0.98	1.0
Parallel-1	0.98	1.0
Aligned-2	N/A	1.0
Smaller-2	N/A	1.0
NextTo-2	N/A	1.0
Parallel-2	N/A	1.0

(a) Interpretation Strengths for Sketch in Figure 5-3(b)

Name	$P[Shape\|stroke\ data]$ after 3 strokes
Wire-1	0.42
Wire-2	0.37
Wire-3	0.01
Battery-1	0.91
Battery-2	0.0
Ground-1	0.03
Line-1	0.99
Line-2	0.92
Line-3	0.03
Aligned-1	0.95
Smaller-1	0.96
NextTo-1	1.0
Parallel-1	0.96
Aligned-2	0.51
Smaller-2	0.62
NextTo-2	0.0
Parallel-2	0.62

(b) Interpretation Strengths for Sketch in Figure 5-3(c)

Table 5.4: Posterior probabilities for the network in Figure 5.4 for the sketch in Figure 5-3.

compare performance of the two hypothesis generation algorithms (slot-based and constraint-based) presented in Section 4.2.

5.5.1 Recognition Accuracy

We measure recognition performance for each system by determining the number of correctly identified objects in each sketch (Table 5.5 and Table 5.7). Each table lists the baseline system performance and the performance of both the Slot (SR-Slot) and Constraint (SR-Con) hypothesis generation algorithms in SketchREAD. In all cases, both hypothesis generation algorithms significantly outperform the baseline system ($p \ll 0.001$); however, the difference in performance between the two methods is not significant.

For the family-tree diagrams SketchREAD performs consistently and notably better than our baseline system. On average, the baseline system correctly identifies 50% of the symbols while SketchREAD-Slot correctly identified 77%, a 54% reduction in the number of recognition errors. Due to inaccurate low-level recognition, the baseline system performed quite poorly on some of

	Size	#Shapes	% Correct		
			BL	SR-Slot	SR-Con
Mean	**50**	**34**	**50**	**77**	**73**
S1	24	16	75	100	94
S2	28	16	75	87	100
S3	29	23	57	78	72
S4	32	22	31	82	77
S5	38	31	54	87	74
S6	48	36	58	78	91
S7	51	43	26	72	72
S8	64	43	49	74	60
S9	84	49	42	61	59
S10	102	60	57	80	72

Table 5.5: Recognition rates for the baseline system (BL) and the two hypothesis generation algorithms in SketchREAD: Slot (SR-Slot) and Constraint (SR-Con). The size column indicates the number of strokes in each sketch.

	Total	% Correct		
		BL	SR-Slot	SR-Con
Female	66	94	97	94
Male	71	52	76	75
Child	170	32	73	66
Marriage	29	52	69	66
Divorce	3	0	0	0

Table 5.6: Recognition rates by shape.

the sketches. Improving low level recognition would improve recognition results for both systems; however, SketchREAD reduced the error rate by approximately 50% on each example regardless of the performance of the baseline system on a particular example. Because it is impossible to build a perfect low-level recognizer, SketchREAD's ability to correct low-level errors will always be important.

Table 5.6 reveals that SketchREAD has the greatest improvement for males (quadrilaterals) and child-links (arrows). This trend arises because the most common error made by the low-level recognizer is to mis-segment poly-lines. SketchREAD is able to re-segment these lines successfully by considering the context in which they appear.

Circuit diagrams present SketchREAD with more of a challenge for several reasons. First, there are more shapes in the circuit diagram domain and the shapes are more complex. Second, there is a stronger degree of overlap between shapes in the circuit diagrams. For example, it can be difficult to distinguish between a capacitor and a battery. As another example, a ground symbol contains

	Total	% Correct			# False Pos		
		BL	SR-Slot	SR-Con	BL	SR-Slot	SR-Con
AC Source	4	100	100	100	35	38	29
Battery	96	60	56	89	56	45	71
Capacitor	39	56	74	69	27	32	14
Wire	1182	62	69	67	478	450	372
Ground	98	18	53	55	0	5	5
Resistor	330	51	52	53	7	7	8
Voltage Source	43	2	37	47	1	6	8
Diode	77	22	15	17	0	0	0
Current Source	44	7	10	11	0	0	0
Transistor	43	0	7	7	0	16	14

Table 5.7: Aggregate recognition rates for the baseline system (BL) and SketchREAD (SR) for the circuit diagrams by shape.

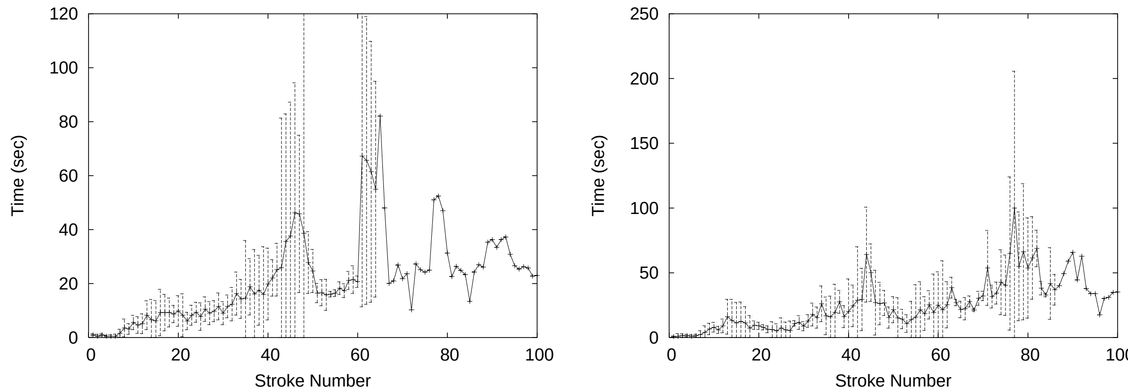
within it (at least one) battery symbol. Finally, there is more variation in the way people draw circuit diagrams, and their sketches are messier, causing the low-level recognizer to fail more often. They tend to include more spurious lines and over-tracings.

Overall, SketchREAD correctly identifies 62% of the shapes in the circuit diagrams, a 17% reduction in error compared to the baseline system. It is unable to handle more complex shapes, such as transistors, because it often fails to generate the correct mapping between strokes and pieces of the template. Although the system attempts to shuffle subshapes in a template in response to new input, for the sake of time it cannot consider all possible mappings of strokes to templates. We discuss below how we might extend SketchREAD to improve its performance on complex domains such as circuit diagrams.

Finally, although the difference in recognition performance between the two hypothesis generation methods is not significant, it is notable that they produce different results at all. The reason for this difference will be discussed below.

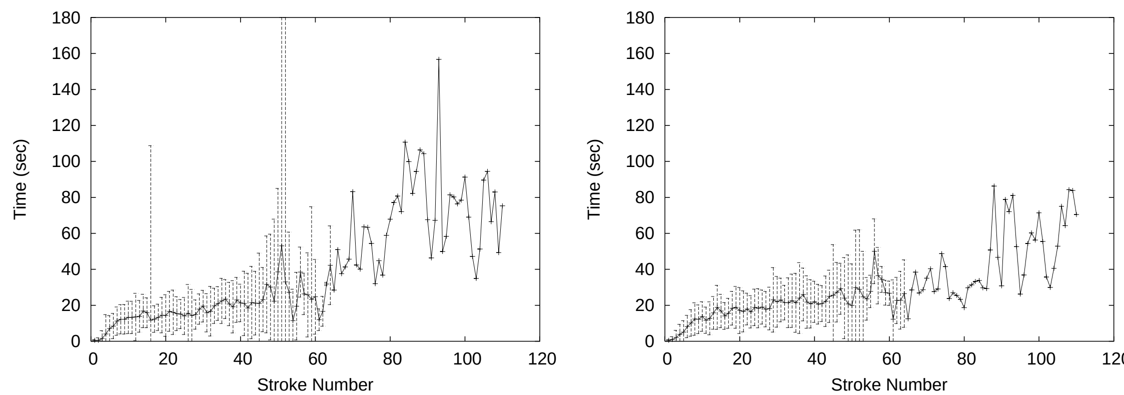
5.5.2 Running Time

We also examined SketchREAD's running time to determine how it scales with the number of strokes in the sketch. Figures 5-5 and 5-6 graph the processing time for each stroke added to each sketch for each domain. In Figure 5-5(a), one family-tree diagram takes a particularly long time to process; that diagram has been omitted from this graph. Three things about these graphs are important. First, although SketchREAD does not yet run in real time, the time to process each stroke in general increases only slightly as the sketch gets larger. Second, the spikes in the graphs



(a) SketchREAD-Slot (b) SketchREAD-Constraint

Figure 5-5: The median incremental time it takes the system to process each stroke in the family-tree diagrams for each hypothesis generation algorithm. Vertical lines indicate standard deviation.

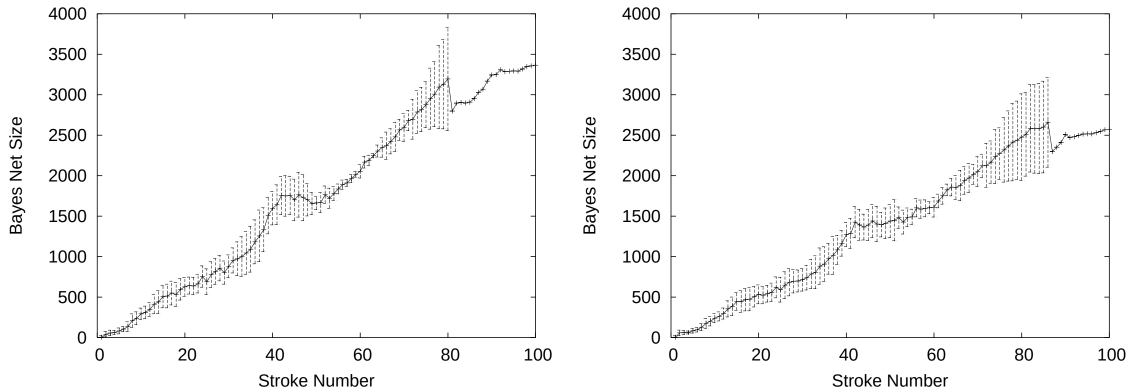


(a) SketchREAD-Slot (b) SketchREAD-Constraint

Figure 5-6: The median incremental time it takes the system to process each stroke in the circuit diagrams for each hypothesis generation algorithm. Vertical lines indicate standard deviation.

indicate that the system takes longer to process some strokes than others. Finally, the processing time for the circuit diagrams is longer than the processing time for the family trees, although this difference is not statistically significant.

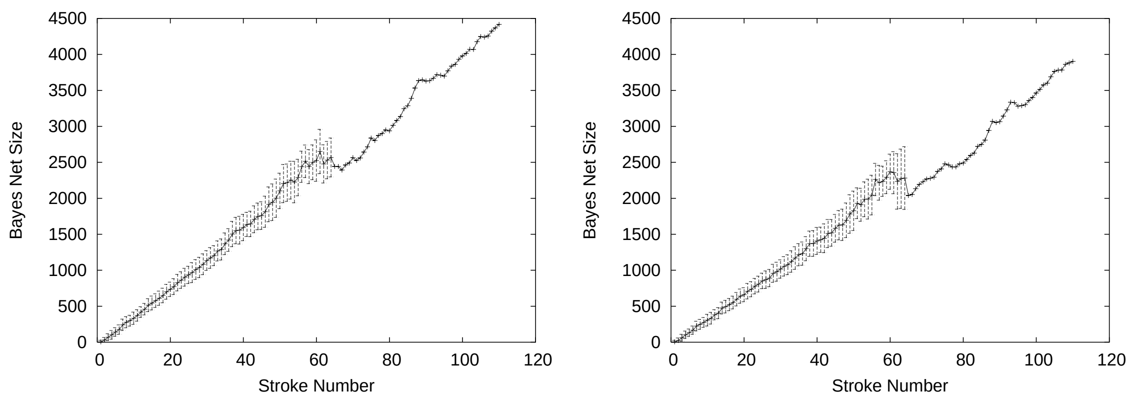
By instrumenting the system, we determined that processing time is dominated by the inference in the Bayesian network, and that all of the above phenomena can be explained by examining the size and complexity of the interpretation network. The number of nodes in the interpretation network grows approximately linearly as the number of strokes increases (Figures 5-7 and 5-8). This result is encouraging, as the network would grow exponentially using a naive approach to hypothesis generation. The increase in graph size accounts for the slight increase in processing time in both graphs. The spikes in the graphs can be explained by the fact that some strokes not only increase the



(a) SketchREAD-Slot (b) SketchREAD-Constraint

Figure 5-7: The median size of the Bayesian network when processing each stroke in the family-tree diagrams. Vertical lines indicate standard deviation. Vertical bars are absent after stroke 80 because there was only one family-tree diagram with more than 80 strokes.

.



(a) SketchREAD-Slot (b) SketchREAD-Constraint

Figure 5-8: The median size of the Bayesian network when processing each stroke in the family-tree diagrams. Vertical lines indicate standard deviation.

size of the network, but have more higher-level interpretations, creating more fully connected graph structures, which causes an exponential increase in inference time. After being evaluated, most of these high-level hypotheses are immediately pruned, accounting for the sharp drop in processing time on the next stroke. Finally, the fact that circuits take longer to process than family trees is related to the relative complexity of the shapes in the domain. There are more shapes in the circuit diagram domain and they are more complex, so the system must consider more interpretations for the user's strokes, resulting in larger and more connected Bayesian networks.

We find that Bayesian network inference takes longer using the constraint-based hypothesis generation algorithm than using the slot-based algorithm. For the family trees, the difference in total processing time is not significant, but the difference in inference time was (Wilcoxon Signed-Rank Test, $p < 0.01$). For circuit diagrams, the Bayesian network inference accounts for a larger proportion of the total processing time in both the constraint-based and slot-based hypothesis generation cases. The difference in total processing time as well as inference time between the two systems is also significant ($p < 0.001$). This result implies that the Constraint-based algorithm produces less fully connected graphs, a point we discuss below.

5.6 Discussion

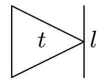
In this section we discuss SketchREAD's limitations and how they can be addressed so that its overall performance can be improved to better handle more complicated domains. We discuss how to improve the system's recognition performance and running time, as well as how SketchREAD compares to a domain-specific recognition engine.

5.6.1 Recognition Performance

SketchREAD significantly improves the recognition performance of unconstrained sketches. However, its accuracy, especially for complicated sketches and domains, is still too low to be practical in most cases. We consider six major sources of recognition error and how they might be addressed to improve the system's performance.

The first source of recognition error comes from the fact that SketchREAD does not distinguish between compatible and incompatible higher-level interpretations for a shape. In general, higher-level interpretations for a shapes are incompatible. For example, in the family-tree domain, a line may be part of an arrow or part of a quadrilateral, but not both. However, some domain patterns are compatible—a female may be part of a marriage and any number of mother-daughter relationships at the same time. Our representation does not distinguish between the two cases. Above we saw that the "explaining away" behavior of the Bayesian network allowed the system to weight competing hypotheses correctly. This same behavior causes the system to weight compatible interpretations incorrectly. For the system to behave correctly, we need to introduce a different Bayesian network structure for compatible relationships that allows them to coexist or even mutually reinforce one another rather than compete with one another.

The second source of error is related to limitation with our constraint-based recognition method. SketchREAD has trouble recognizing shapes that constrain the subcomponents of their subshapes.

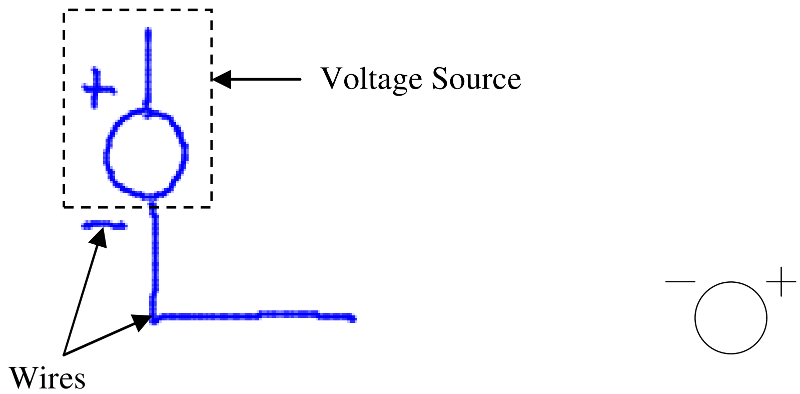


DEFINE ARROW
(**Subshapes**
(Line l)
(Triangle t))
(**Constraints**
(touches $t.l_2.p_2$ l)
(equalLength $t.l_1$ l)
(parallel $t.l_1$ l))

Figure 5-9: The description of a diode.

For example, the description of a diode (Figure 5-9) includes the constraint that point p_2 of l_2 of triangle t must touch line l. In isolation, the lines in a triangle are all the same, and the system arbitrarily labels them l_1, l_2 and l_3. If the labels are not placed correctly, the system will not be able to recognize the diode symbol. This problem arises from the fact that the system assigns labels when a shape is created, rather than when that shape is fit into a template for a higher-level shape. Dynamically assigning labels to subcomponents is a non-trivial problem that we do not yet handle. Because the problem of subcomponent labeling causes such a problem for lines, we handle them as a special case. When the system recognizes a line, it arbitrarily assigns a labeling of p_1 and p_2 to the endpoints. Because this labeling is likely to be incorrect, the system instantiates a second line interpretation with the opposite endpoint labeling. This policy aids recognition, but often results in a number of redundant interpretations for the user's sketch.

The third source of error is due to the lack of sufficient context provided by the descriptions for each domain. On a small scale, domain patterns for the circuit domain could have prevented false positives and aided recognition. Errors such as the one in Figure 5-10, in which the system incorrectly but reasonably chose the wire above the voltage source as the minus sign in the voltage source and labeled the true minus sign as a wire, could have been prevented if the system knew that voltage sources must be attached to wires on both sides. On a larger scale, the system is also lacking more global context that could aid recognition. For example, in the family-tree diagrams, each person is allowed to be involved in only one marriage relationship, but can be involved in any number of parent-child relationships.



(a) SketchREAD's incorrect labeling

(b) The voltage source symbol

Figure 5-10: An incorrectly recognized voltage source symbol.

A fourth source of error arises from the necessary limitation on the number of higher-level interpretations a shape may have. To control the complexity of the Bayesian network, we limit the number of parents a node in the graph may have (i.e., we limit the number of higher level interpretations for each shape). This limitation causes a less fully connected graph structure, measured by determining the size of the largest clique produced using the junction tree inference algorithm [52, 43]. Occasionally, there are too many high level interpretations to consider and some must be ignored. Running time improvements suggested below could allow the system to consider more higher level interpretations in a reasonable amount of time.

A fifth source of recognition error has to do with the fact that our recognition algorithm treats every stroke the user draws as a meaningful piece of a symbol, which is not always the case. Spurious lines and over-tracing hindered the system's performance in both accuracy and running time. A preprocessing step to merge strokes into single lines would likely greatly improve the system's performance. Also, in the circuit diagram domain, users often drew more than one object with a single stroke. A preprocessing step could help the system segment strokes into individual objects.

Finally, while SketchREAD always corrected some low-level interpretation errors, its overall performance still depended on the quality of the low-level recognition. Our low-level recognizer was highly variable and could not cope with some users' drawing styles. In particular, it often missed corners of polylines, particularly for symbols such as resistors. Other members of our group are working on a low-level recognizer that adapts to different users' drawing styles.

5.6.2 Processing Time

In general SketchREAD's processing time scaled well as the number of strokes increased, but it occasionally ran for a long period. The system had particular trouble with areas of the sketch that involved many strokes drawn close together in time and space and with domains that involve more complicated or overlapping symbols. This increase in processing time was due almost entirely to increase in Bayesian network complexity.

We suggest two possible solutions. First, part of the complexity arises because the system tries to combine new strokes with low-level interpretations that are already part of complete, correct high-level interpretations. These new interpretations are pruned immediately, but they temporarily increase the size and complexity of the network, causing the bottlenecks noted above. In response, we are testing methods for "confirming" older interpretations and removing their subparts from consideration in other higher-level interpretations. The system would also confirm their value as *true* in the Bayesian network so that their posterior probabilities do not have to be re-computed.

Second, we can modify the system's inference algorithm. We experimented with several inference algorithms and found that Loopy Belief Propagation was the only one that gave accurate results in a reasonable amount of time. Loopy Belief Propagation is an inference algorithm that repeatedly sends messages between the nodes until each node has reached a stable value. Each time the system evaluates the graph, it resets the initial messages to 1, essentially erasing the work that was done the last time inference was performed, even though most of the graph remains largely unchanged. Instead, this algorithm should begin by passing the messages it passed at the end of the previous inference step.

Next we compare the running time between the two hypothesis generation methods. We found that the Constraint method took considerably less time on inference, suggesting that it produced simpler interpretation graphs. This result was unexpected because we anticipated that the main savings of the Constraint algorithm would be in the time it took to generate the hypotheses. However, it appears that the Constraint algorithm actually produces fewer overlapping higher level interpretations for lower-level interpretations, resulting in a simpler interpretation graph. This result can be explained by the fact that our threshold for fitting shapes into template in the Slot algorithm was more liberal than the method the constraint algorithm was using to generate hypothesis. That both methods performed with similar recognition performance implies that the thresholding method used in the Slot algorithm is too liberal.

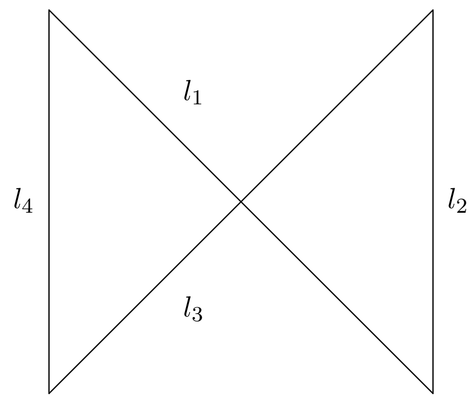


Figure 5-11: Four lines that meet our under-constrained description of a quadrilateral but that we did not intend to recognize.

5.6.3 Challenges of a Multi-domain Recognition System

One advantage of SketchREAD over a domain-specific system is that it does not require programming in order to be applied to a new domain. On the other hand, writing shape descriptions is not always trivial. We were surprised at the number of false positives our system produced and at the difficulty we faced in writing domain descriptions that were constrained enough to prevent these false positives. In particular for circuit diagrams, we modified our descriptions several times, adding more constraints each time. Originally, we hypothesized that our shape descriptions could be slightly under-constrained because people would not draw shapes that differed significantly from the correct description. However, we did not anticipate that the system would encounter a number of shapes that inadvertently satisfied those under-constrained descriptions, shapes that the user did not intend to produce, but ended up creating in the process of drawing all of the shapes together in one diagram. For example, we did not include the constraint that the lines in a quadrilateral should not cross, although we intended for this constraint to hold. Users never produced quadrilaterals such as the one in Figure 5-11 on purpose, but this shape appears several times among the shafts of the arrows in the top-right sketch in Figure 5-1(a). The difficulty we faced in creating effective shape descriptions highlights the importance of a system to help people develop their shape descriptions or an automatic system to learn them, such as those being developed by Hammond [35] and Veselova [76].

The main advantage of using a domain-specific system is in the efficiency of searching for and generating hypotheses. If we can write specific recognizers for each shape, we can tailor them to the patterns users typically use when drawing that shape. For example, for certain shapes, people have a preferred drawing style or order. This natural drawing order should be exploited in a multi-domain system to help the system constrain the search for possible hypotheses for the user's strokes. This idea is being explored by Sezgin [70].

5.7 Summary

We have demonstrated how our system performs on real-world data in two non-trivial domains. We find that it was successfully able to reinterpret low-level missed interpretations to improve performance over a baseline system. We also illustrated that the semantics of our model have promise for incorporating context into recognition and allowing higher level information to influence low-level interpretations to resolve ambiguity in the drawing.

Our goal is to use SketchREAD in sketch recognition user interfaces to explore the human computer interaction issues involved in creating such interfaces. Although SketchREAD is not yet powerful enough to handle unconstrained sketches in complicated domains, for simple domains it performs accurately in near real time. In the next chapter how it was used in an implemented sketch recognition user interface and we explore user interface issues for a recognition-based digram creation tool.

Chapter 6

Building Sketch Recognition User Interfaces

To date, sketch recognition has not been reliable enough to use in pen-based interfaces. Most current pen-based computer design tools explicitly avoid recognizing the user's strokes. Our long-term goal is to make sketch recognition reliable enough that it can be incorporated into sketch-based early stage design tools for a wide range of domains. We call this class of systems *Sketch Recognition User Interfaces* or *SkRUIs* to emphasize that they are not merely pen-based tools, but also recognition-based tools.

Building a robust recognition engine is only one of the challenges in building a usable SkRUI. We face a number of challenges in incorporating free sketch recognition into a design tool, including when to display recognition feedback, how to integrate sketching with editing, and how to allow the user to correct recognition errors. SkRUIs are fundamentally user interfaces, and to study the above issues, we need to build and test such an interface with actual users.

Although not ready to be used in SkRUIs for complex domains, SketchREAD may currently be used in simple domains where the shapes have little overlap and are drawn spatially separated. In keeping with this observation, we used SketchREAD to build a sketch-based system for constructing relationship diagrams in PowerPoint. This tool recognizes naturally drawn diagrams and automatically imports them into PowerPoint. It combines the ease of drawing on paper with PowerPoint's sophisticated presentation capabilities.

SketchREAD produced few errors in this simple domain, enabling us to examine a number of usability challenges of incorporating free-sketch recognition into a diagram creation tool. We present guidelines for creating this type of Sketch Recognition-based User Interface based on a series of informal user studies with a prototype implementation. These guidelines take into consideration both the requirements needed to maintain robust sketch recognition and what felt natural to users.

6.1 Application

The interface design is shown in Figure 6-1. Our sketch recognition application communicates with PowerPoint, but runs in a separate window. The user sketches diagrams directly onto the slide in the center of the window. The slide is automatically updated to reflect the slide currently showing in PowerPoint, and the objects sketched are automatically added to the PowerPoint slide when the user switches back to the PowerPoint window. The user may edit the diagram either by entering edit mode using the toggle buttons at the top of the window, or by entering "online" edit mode by holding down the pen until the cursor changes to an arrow (when the user lifts the pen the system returns to sketch mode). The user may edit all shapes in the diagram, even if she did not originally sketch them.

The system supports both recognized drawing, where the user's strokes become clean PowerPoint objects, and unrecognized drawing, where the user's strokes appear on the slide exactly as they were drawn.

6.2 System Evaluation

Throughout the design process, we conducted formative evaluation to guide the system's design. The goal of this evaluation was to understand users' perceptions of our tool and what they wanted from such a tool, rather than to prove that we had created an effective system.

We evaluated our interface with three users on three prototypical tasks involving diagram creation. Each was a common task performed during diagram creation and was selected to explore a specific aspect of interacting with the system. The first task explored how the user used the diagram creation tool to create a new drawing, beginning with the task of creating a new slide, which is done using the menu. The second task explored the interaction between the pen-based creation tool and keyboard text entry and editing: We asked users to create a diagram and then label it using PowerPoint. Our system does not support handwriting recognition, and we wanted to see if this presented

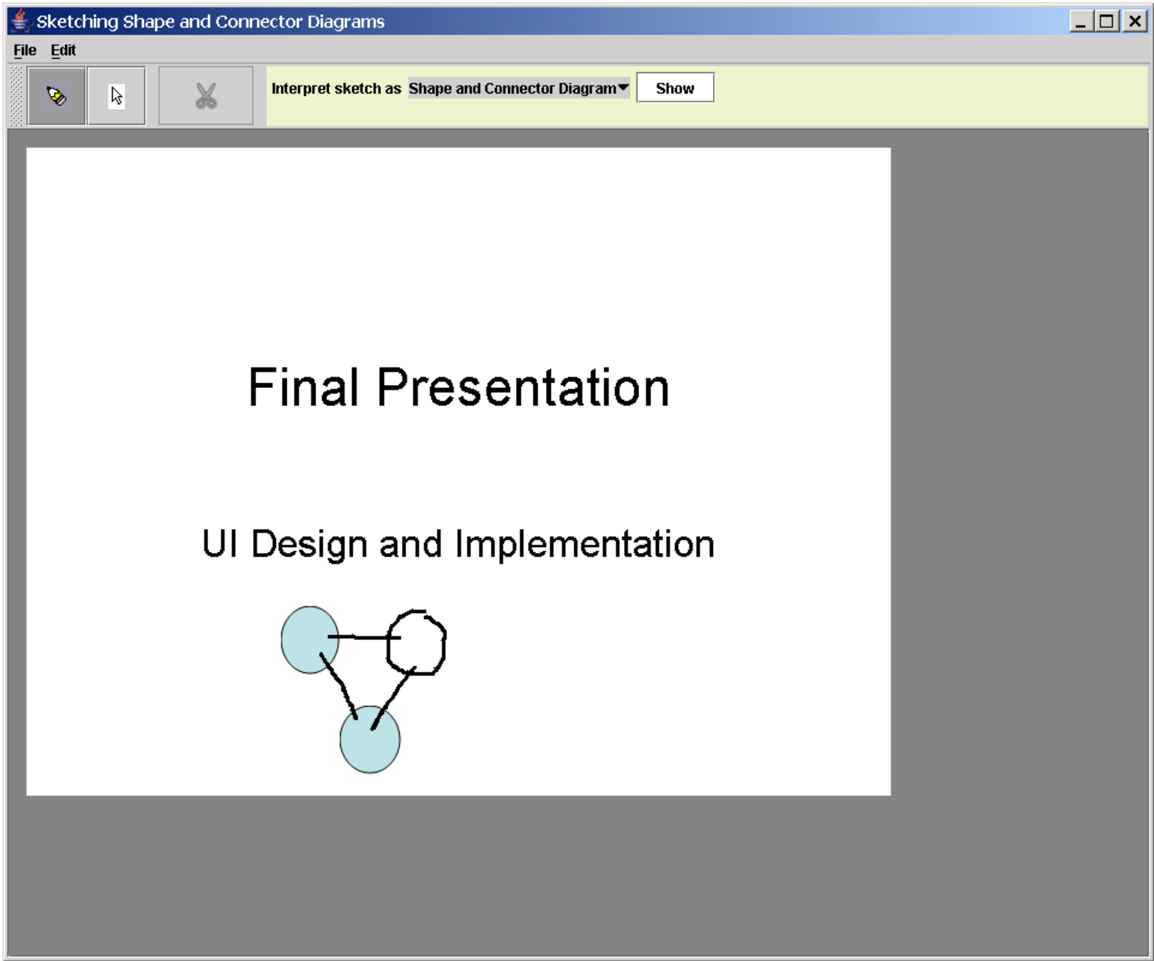


Figure 6-1: The final design of our PowerPoint diagram creation tool.

a problem or if users felt comfortable using the keyboard to enter text. The third task explored the interaction between sketching and editing using the pen. We asked the user to sketch a diagram and then move and delete pieces of the diagram and then continue to draw.

We used several evaluation methods throughout the design process. Our design/evaluation stages included a paper prototype tested with users, a low-fidelity computer prototype both tested with users and evaluated using heuristics, and the final system tested with users.

6.3 Design Guidelines

We give guidelines and advice for incorporating sketch recognition into diagram creation tools based on this design/evaluation process and our knowledge of the requirements for robust sketch recognition.

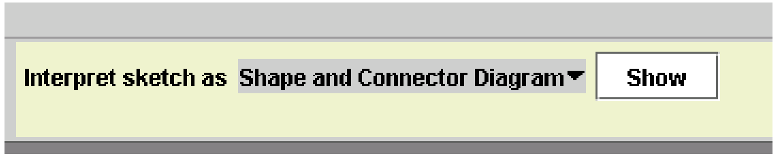


Figure 6-2: A close-up of the PowerPoint diagram creation tool.

Display recognition results when the sketch is complete

The system should display recognition results after a user has completed a diagram (or piece of a diagram), rather than after every stroke. Previous work has suggested that recognition feedback can distract the user during the design task [40]; we have observed this effect in previous work [3]. In response, our diagram creation interface did not display recognition feedback while users sketched. We found that users sketched their diagrams without pausing between shapes and did not appear to expect recognition feedback. Only one user paused after drawing his first stroke, expecting that stroke to be recognized, and he acknowledged that his expectations were based on his interaction with our previous recognition interface.

In determining when to display recognition results, our observations suggest that a SkRUI can should rely on explicit cues to determine when to display recognition results. A difficult problem in determining when to display recognition results is determining when the user is done sketching. Implicit cues such as how long the user has paused since her last stroke are not always reliable. Particularly in design tasks, users may pause in the middle of the task, but do not want to receive recognition feedback at this point. In our evaluation, we found window focus was a reliable indicator that the user had completed a sketch because of the fact that when user's had completed a sketch, they usually switched back to the PowerPoint window. Users also clicked the "Show" button at the top of the window (Figure 6-2). Both methods of indicating they were done drawing did not seem to inconvenience the user in their tasks. We conclude that asking the user to explicitly inform the system when a diagram is complete (or at least partially complete) is a viable option for a SkRUI.

Provide obvious indications to distinguish free sketching from recognition

Allowing the user to draw strokes that remain unrecognized by the system presents a challenge for a SkRUI. For example, a user might want to add an informal, sketched annotation next to a clean recognized diagram. Because SketchREAD recognizes freely drawn sketches, where symbols can be composed of more than one stroke, it is difficult to determine when the user wants her strokes recognized and when she wants them to remain unrecognized. To support this functionality, we provide explicit “free sketch” and “recognition” modes. The combo box at the top of the window (Figure 6-2, currently set to “Shape and Connector diagram”) indicates the system’s current recognition mode. We found that users could understand the difference between the modes, but many users forgot what mode they were in, despite the label at the top of the window. One user drew an entire digram in free-sketch mode and could not figure out why the system failed to recognize his strokes. The system needs to use more obvious clues (such as changing the cursor or the stroke style, perhaps) to help users remain aware of what recognition mode they are in.

Restrict recognition to a single domain until automatic domain detection becomes feasible

Our application supports the recognition of only one domain: relationship diagrams. We would eventually like to build a tool that supports the recognition of more than one domain. We experimented with such a system in our early paper prototypes, asking users to draw both relationship diagrams and circuit diagrams. To recognize the user’s strokes, SketchREAD must know in which domain the user is drawing. Automatic domain detection represents a technical challenge that our system cannot yet handle, so we asked users to explicitly choose the domain. Despite this request, users tended to attribute universal recognition capabilities to the system and did not understand that they had to specify a particular recognition domain. For example, we asked users to draw a relationship diagram right after they had drawn a circuit diagram. If they failed to change the recognition domain, we indicated that their sketch had been incorrectly recognized as a (messy) circuit diagram. None of our three users performing this task correctly switched domains, and none of them could figure out what had gone wrong. We conclude that single-domain tools will be most effective until automatic domain detection becomes a feasible alternative, or until we explore more effective ways to help the user understand the system’s underlying recognition model.

Incorporate pen-based editing

The system should incorporate as much editing capability as possible (including copy, paste, alignment, resizing, etc.) into a sketch-based diagram creation interface because users wanted to simultaneously create and clean up their drawings. In our final testing, one user repeatedly redrew a circle until it was the exact same size as another on the screen. After drawing the circle she carefully aligned it with the two she had already drawn. The ability to copy, paste, and align her objects could have saved her a considerable amount of effort. Although this behavior may not be as prevalent for rough design tasks, based on previous work [3], we anticipate that users will still want the ability to copy and paste parts of their drawings.

Sketching and editing should use distinct pen motions

The system should the system should support pen-based editing gestures that are distinct from any pen movement the user would make while sketching. These gestures allow for the user to avoid explicitly specifying sketching and editing mode, but should be specifically designed to be less prone to mis-recognition in a free-sketch system.

SkRUIs require large buttons

When given the option of buttons, menus, and keyboard shortcuts, users chose to use the button with few exceptions. Furthermore, these buttons should be larger than traditional buttons because of the difficulties in targeting with the pen. In our initial tests, users could not reliably click buttons that were the same size as those designed for a mouse-based interface. When we enlarged the button-size they had no trouble.

The pen must always respond in real time

We are trying to build a system that is as natural as sketching on paper. To do so, the system must not allow recognition to interfere with the user's drawing. It is acceptable that recognition not occur in exactly real time; users tolerated a slight recognition delay once they were done drawing. However, it is extremely important that the computational requirements of the recognition system not overwhelm the system's ability to display the user's strokes exactly as they are drawn. In our tests, when the user's strokes did not appear when the user's tried to sketch them, users became agitated, repeatedly drawing the same shape until it appeared under their pen as they drew. These added strokes confused the recognition system, causing it to add extra shapes to the user's diagrams. Because the users thought they had drawn a single shape, they could not understand the existence of the extra shapes in the diagram.

6.4 Extensions

In our simple domain, SketchREAD rarely produced errors. However, in general SketchREAD will never eliminate recognition error, and the user will have to correct the system's errors. Sketch-

READ's interpretation-graph architecture could itself be used to reduce the burden placed on the user in correcting the system's errors. Often in complicated diagrams, errors are interconnected: If there are many competing interpretations, choosing the wrong interpretation for one part of the diagram often will lead the system to choose the wrong interpretation for the surrounding strokes. SketchREAD could display its recognition results on top of the user's original strokes, so that the user can see the context of these results. Then, the user could help the system by tracing over the strokes for one of the symbols that was mis-recognized. This retracing would help the system recover from the error because the user's strokes would be cleaner and because the system would know that they were all part of a single symbol. Then, based on this new interpretation, the system could reevaluate the surrounding strokes and (hopefully) recover some of the missed interpretations that might still exist but simply were not currently chosen as the best interpretation.

One of the main causes of recognition error might be dealt with through UI design. The system had trouble with the fact that the users varied the structure of their symbols even though they were explicitly shown the desired structure for each symbol. For example, although we instructed people to draw a voltage source using a circle with a plus and a minus next to it, some people put the plus and minus inside the circle. SketchREAD is designed to handle variations in the way people draw, but cannot handle such fundamental changes to the shape of the symbol. Although ideally we would like to support all methods people have for drawing each object, this might never be possible. Instead, a simple interactive training step before a new user uses the interface could help eliminate this type of variation without imposing too many limitations on the user's drawing style.

The diagram creation tool presented here is an early prototype with limited functionality, not yet powerful enough for deployment or more formal user studies. We aim to extend its functionality and conduct a more formal comparison between this system and PowerPoint to help us better understand the utility of our system. A more formal evaluation will also help us refine the design and evaluation guidelines presented here.

Chapter 7

Related Work

With the rise of pen-based technologies, the number of sketch-based computer tools is increasing. Sketch-based computer systems have been developed for a variety of domains including mechanical engineering [4, 33, 73], user interface design [11, 48, 63], military course of action diagrams [14, 22], mathematical equations [61], musical notation [8, 23], software design [36, 49], and image editing [69]. In addition, a few multi-domain recognition toolkits have been proposed [50, 56].

It is important to recognize that building a sketch-based system is not necessarily the same as building a sketch recognition system. The focus of many of the systems above (including [11, 22, 48, 63, 69]) was on creating a good user interface—a worthwhile and challenging task in itself. Rather than trying to tackle the problem of sketch understanding, these researchers designed interfaces that would help users accomplish sketch-based tasks, but provided little free-sketch recognition. This dissertation, by contrast, presents a new recognition technique rather than an in-depth exploration of user interface issues. Accordingly, this chapter focuses on previous algorithms for online sketch recognition as well as algorithms for related tasks including speech recognition, diagram recognition, and computer vision.

7.1 Online Sketch Recognition

Sketch recognition algorithms can be divided into three categories: *sketch beautification*, *gesture recognition*, and *continuous sketch recognition*. Sketch beautification focuses on cleaning up roughly drawn strokes, while the other two categories focus on providing meaning for the user's strokes. The core difference between gesture and continuous sketch recognition is that gesture recognition deals with only classification given a set of strokes, while continuous sketch recognition simultaneously deals with segmentation and classification.

7.1.1 Sketch Beautification

Hand drawn diagrams are easy to produce, but too messy for most uses. Sketch beautification focuses on cleaning up hand drawn strokes.

Although often associated with recognition, beautification is largely orthogonal to recognition and can be used with or without recognizing the symbols in the user's diagrams. Sketch beautification systems such as Pegasus [41] and Easel [42] do no symbolic recognition, relying instead on common geometric constraints to tidy up the user's strokes as they are drawn, allowing users to create formal diagrams quickly and naturally. When beautification is used in conjunction with symbol recognition, it can provide a natural way for a sketch system to convey its interpretation of the diagram to the user, as in [5] and [6].

7.1.2 Gesture Recognition

Just as speech recognition began with isolated word recognizers, sketch recognition began by focusing on gesture recognition. Approaches have been developed for both single stroke and multi-stoke gesture recognition.

A large body of work focuses on single-stroke gesture, or *glyph*, recognition. Early work in this area was done by Rubine [68]. His system recognized a set of glyphs through the use of stroke features (e.g., the area of the bounding box) rather than the raw ink. More recent work by Long *et al.* [54] notes that the careful design of gestures can further improve recognition. He has developed a system that helps users design a set of gesture commands that will not be confused easily by the computer [55]. Work by Sezgin *et al.* demonstrated a single-stroke segmentation system that found the corners in a user's stroke and classified the stroke either as a line, an ellipse, or a set of lines or curves (or a combination) [71]. These basic shapes then can be used to form more complex objects.

Single-stroke classification is useful for some tasks and can be an invaluable subcomponent of a sketch-based design tools, but used on its own it suffers from two main limitations. First, by definition, users must draw each symbol with a single stroke, often using a pre-specified drawing style (e.g., a horizontal line may only be recognized if it is drawn from left to right). Second, these recognizers deal with each stroke in isolation and do not incorporate the context in which the stroke was drawn.

The Quickset system extends the notion of gesture recognition to multiple strokes. Given a set of strokes known to form a single symbol, Quickset recognizes a pre-segmented set of strokes as a military course of action diagram symbol, using multi-modal information (speech) to improve recognition performance. Quickset uses a multi-tiered recognition architecture to integrate speech input with drawing input, allowing these noisy signals to mutually disambiguate one another. Oviatt *et al.* have shown that this mutual disambiguation can improve recognition [64].

Because of its segmentation assumptions, gesture recognition, both for single strokes and multiple strokes, has become robust enough to be integrated into usable sketch-based design tools. A number of existing sketch systems use mainly gesture recognition rather than continuous sketch recognition to process the user's strokes [14, 17, 23].

7.1.3 Continuous Sketch Recognition

Continuous sketch recognition involves segmenting and recognizing the user's strokes simultaneously. The process of finding the correct stroke groupings can take exponential time in the worst case. This running-time bottleneck has led researchers to make a number of limiting (and often faulty) assumptions about how the sketch will be drawn to aid segmentation.

A number of online sketch recognition systems rely on temporal constraints to prune the search space. The UML system developed by Lank *et al.* assumes that objects are drawn using consecutive strokes [49]. Other systems limit the search space even more by employing a greedy recognition strategy—once a high-level shape (e.g., an open-headed arrow) is identified, its low-level shapes (e.g., the lines) are not considered in any future high-level interpretations [63, 78, 80]. While this approach does not limit objects to being drawn with consecutive strokes, it creates undesired interdependencies within object descriptions. For example, the shapes in UML include both an open-headed arrow and a diamond-headed arrow. In drawing a diamond-headed arrow, the user might first draw an open-headed arrow and then close the diamond head. However, because the system immediately recognizes the open-headed arrow, the diamond-headed arrow would not be recognized unless it had previously been defined as an open-headed arrow with two additional lines.

Recent work by Sezgin *et al.* relies on preferred stroke orders to aid recognition [70]. He showed that when asked to draw a symbol more than once, people tended to draw it in the same order and that certain regularities exist between individuals. He used these to construct a Hidden Markov Model based on these orders for recognizing each symbol. Because not all stroke orders can be modeled, he suggests that his model be used to compliment a shape-based model such as the one presented here.

Many online sketch recognition systems use spatial constraints to limit their search. Spatial constraints can be used in two ways. First, most systems consider only groups of strokes that fall within a spatial bound (e.g. [4, 36, 72]). This spatial bound on its own may not be enough, especially for regions that contain many overlapping strokes. For example, if there are ten lines that all fall within a small region, to identify an arrow, the system still may have to try $(10 \cdot 9 \cdot 8)$ combinations of these lines. To further limit the search space, many systems set a hard threshold on constraints between subshapes (e.g., two lines connect if their endpoints are less than ten pixels apart) [4, 48, 49, 50, 57, 78]. Using these thresholds in the arrow example above, the system would only have to consider sets of lines whose endpoints were connected. Finding these connections would require searching near only $10 \cdot 2 = 20$ line endpoints and would result in considerably fewer combinations to consider. Unfortunately, as we have argued, it is difficult (if not impossible) to set a threshold on these constraints that suitably prunes the search space and does not rule out correct combinations of strokes. Whether or not two lines are part of an arrow depends on whether or not they connect; however, whether or not they appear to connect actually can depend on whether or not they are part of an arrow.

Finally, many on-line sketch recognition systems rely on user feedback to maintain a correct segmentation and interpretation. When a sketch system makes its interpretation of the sketch immediately clear as the user draws, the user can immediately correct any missed interpretations, avoiding confusion later on. However, Hong *et al.* note that the recognition results and the process of keeping the system's understanding of the drawing correct can interfere with the design process [40]. The goal of our system is to recognize a sketch completely in the background, without requiring any feedback from the user.

7.2 Related Recognition Tasks

Although the field of online sketch recognition is relatively small, it is closely related to many other fields. We review related work on the tasks of speech recognition, handwriting recognition, diagram recognition, and computer vision as they relate to online sketch recognition.

7.2.1 Speech and Handwriting Recognition

Our approach to sketch recognition was inspired by early work in speech recognition. The Hearsay-II system used a blackboard system to maintain multiple hierarchical interpretations for a user's utterance [19], relying on the context that the higher levels provided to help disambiguate the signal. While our architecture is similar, we take a more structured approach to modeling the uncertainty throughout the recognition process with the use of Bayesian networks.

Handwriting consists of two-dimensional shapes (letters) produced with a one dimensional signal. Because of the strong temporal component of handwriting, a standard recognition approach models features of a user's stroke in a temporal stream using a Hidden Markov Model or similar time-based model. However, a few approaches use a shape-based approach, modeling two-dimensional relationships independent of time. Work by Burl and Perona uses a shape based representation of characters or partial words and then a hierarchical word model to spot keywords in cursive text [10]. Recent work by Cho and Kim uses a Bayesian network model similar to our model to recognize Hangul Characters based on their shape [13]. The shape models used by these systems are more detailed than our models because of the complex and subtle variations in the shapes of handwritten characters.

7.2.2 Diagram Recognition

Like sketch understanding, diagram recognition also involves recognizing two-dimensional symbols, but diagrams generally do not contain temporal information. On the other hand, compared to sketches, diagrams are relatively clean and free from low-level noise.

Recognizing a diagram (like recognizing a sketch) is a matter of parsing the elements in the diagram according to a specified visual language. A variety of general two-dimensional parsing techniques have been developed for diagram recognition [39, 59, 60]. Given a set of two-dimensional, hierarchical shape descriptions, these systems combine low-level shapes in a diagram according to the constraints given in the shape descriptions to produce higher-level descriptions for the diagram. Of course, when grouping these shapes, it may be possible to combine them in more than one way, leading to ambiguities during parsing. While natural-language systems encounter these same parsing ambiguities, the fact that diagrams are two-dimensional provides even more room for ambiguities.

As in our system, targeted parsing routines can help avoid the combinatoric explosion caused by parsing ambiguities. For example, a system by Kasturi uses targeted segmentation routines to separate text from shapes in diagrams and then to recognize those shapes [45]. Specific routines, such as a dashed-line finder and a routine that specifically detects text that is connected to graphics, help this system avoid infeasible processing times. Too many targeted routines can sacrifice the generality of a system, but some specific low-level processing routines can greatly improve recognition without relying on too many domain assumptions.

7.2.3 Computer Vision

In general, recognizing objects in an image is more difficult than recognizing patterns in a sketch because of the difficulty of segmentation—separating the objects in the image from the background and from one another. In addition, the sketched symbols we aim to recognize are generally simpler and more stylized than objects in the natural world. Nonetheless, both tasks involve two-dimensional object recognition, and here we examine how our approach relates a number of computer vision techniques. For the sake of clarity, we divide our discussion into two (overlapping) categories: techniques that use relational and spatial models, techniques that use graphical models.

Vision using Relational and Spatial Models

One of the fundamental questions in computer vision is how to represent the objects to be recognized, given that the position, orientation and size of the object may vary widely from image to image. There are two general ways to deal with this problem. Data-based approaches represent an object as a set of examples, each in a different size, orientation and position in the image. Recognition involves a direct comparison between each exemplar and a new image to be recognized. Model-based approaches, by contrast, use a single, often abstract, representation of each object to be recognized that can be transformed to match objects in different sizes, positions and orientations. Model-based recognition involves comparing various legal transformations of the model to the image to be recognized.

The abstract nature of many sketched symbols leads to a wide variation in how they are drawn. For example, a quadrilateral must have four sides, but there are no restriction on the relative lengths of or angles between those sides. We would need a prohibitively large amount of data to represent any legal quadrilateral we might encounter, so we focus on model-based approaches which allow us to naturally represent this type of under-constrained symbolic patterns.

The main challenge for model-based recognition is efficiently searching for the correct match between a transformed version of the model and a portion of the image. In the worst case this process involves comparing all possible transformations of a model to each portion of an image—clearly an unfeasibly large search problem. For rigid objects, a number of approaches have been proposed to make this approach more efficient [25]. Here, as our objects are not rigid, we examine a number of techniques for improving the efficiency of this search for models of non-rigid objects.

A relational model is a hierarchical non-rigid object model that represents an object as a set of subshapes and relationships between them. For example, using a relational model a quadrilateral would be represented as four lines with the appropriate connections between the endpoints of those lines. We use a probabilistic version of this model to represent sketched symbols, but, as used in many early computer vision systems [20, 82], these models were originally deterministic.

Matching the relational object model in an image involves verifying that all of the necessary subparts and relationships are present in the image. One effective way to perform this match is to represent the objects and relationships in each model, as well as the low-level shapes and relationships in each image (discovered using low-level recognition routines), as a graph. Then, recognition involves matching the model graphs to the image graphs. This problem, called subgraph isomorphism, is NP-complete, and much work has been devoted to developing efficient matching approaches [2, 15, 18, 21, 62]. A related approach constructs a tree of possible matches model features and image features. Each level in the tree encodes all the possible matches between a single image feature and all possible model features. Each path from the root to the leaves of the tree represents a complete hypothesized match between image features and model features. Because the complete tree is too large to consider in real time, researchers have proposed methods for efficiently pruning the tree without sacrificing interpretation correctness [31].

Deterministic relational models are limited by the fact that there is no systematic way to model the importance of the various subcomponents or relationships that make up a model. Probabilistic variations on relational models address this problem. A face detection approach by Burl and Perona uses a hierarchical generative model to recognize a face in terms of its subparts and incorporates spatial information in the model. They represent a face as two eyes, a nose, and a mouth, that must satisfy specific spatial relations. They use an eye, nose and mouth detector to detect the presence of each low-level facial component and then answer the question, given that they have detected eyes at positions x_1 and x_2, a nose at x_3 and a mouth at x_4, what is the probability that there is a face at position $\mathbf{F}$? In other words, how does

$$P[\text{one face at } \mathbf{F}|X_le = x_1, X_re = x_2, X_m = x_3, X_n = x_4]$$

compare to

$$P[\text{no face}|X_le = x_1, X_re = x_2, X_m = x_3, X_n = x_4]$$

Note that this model is robust to low-level sensor failures (e.g., even if the mouth has not been detected, $P[\text{one face at } \mathbf{F}|X_le = x_1, X_re = x_2, X_n = x_4]$ may still be quite high. Furthermore, unlike in deterministic relational models, it provides a natural way to weight the input of certain sensors higher than others. For example, we might know that our mouth detector produces many false positives. Hence detecting a mouth should not raise the system's belief that there is a face present as much as detecting a nose does.

The fundamental difference between these previous approaches and our approach is in the richness of constraints between subcomponents that can be modeled. Early relational models supported wide variety of constraints, but represented these constraints simply as being present or not present. Recent approaches such as as the face detection system by Burl and Perona allow noisy detectors to have less influence over the overall interpretation of the image, but the relationships that they are able to express are purely spatial. They support positional constraints such as above, or left-of; we generalize the notion of constraint and allow for a richer variety of constraints, for example relative size, or relative orientation while maintaining the advantages of a probabilistic framework.

Vision using Graphical Models

Several other graphical model based approaches to computer vision are closely related to our work.

A number of approaches use graphical models to enforce local consistency between neighboring regions in an object. Although Hidden Markov Models (HMMs) are traditionally used to model one dimensional, time-dependent data, a number of systems have adapted these techniques to recognizing two-dimensional data. A system by Muller *et al.* transform two dimensional features into a one-dimensional input stream by starting at the center of the object and spiraling outward, measuring features such as the percentage of black pixels in a small image block at the given location. Li *et al.* use two dimensional HMMs to model the spatial dependencies between neighboring regions of an object. Both of these approaches work best on pre-segmented regions of an image.

Markov Random Fields (MRFs) are another type of graphical model that can be used to model the dependencies between neighboring regions of an image [27]. In recent work, Coughlan and Ferriera define an MRF where each node in the MRF is a point in the shape to be recognized and the edges in the MRF constrain the positions of these points relative to one other [16]. Their model is successful at recognizing patterns in an image in the face of a high degree of noise and distractor shapes, although it can only be used to locate a single instance of a single shape in an image. Although quite powerful, MRFs are best at describing local dependencies (for example, a straight edge is more likely to continue in the same direction in a neighboring region than to suddenly switch direction). It is not as clear how to use them to encode higher level dependencies (e.g., that there should be four lines in a quadrilateral).

There are a limited number of systems that use hierarchical Bayesian models to represent objects to be recognized. Bienenstock proposes a theoretical formulation for composing low-level objects into higher-level groupings inspired by the task of text recognition [7]. Westling and Davis [81], Kumar and Desai [47] and Liang et. al. [53] all propose general, hierarchical Bayesian network based approaches to object recognition that are very similar to our own approach. Unfortunately, how these models fare when used for recognition is not well documented and further evaluation is needed to illustrate their strengths and weaknesses in practice.

Recent work by Wachsmuth and Sagerer uses a dynamically constructed Bayesian network architecture to match verbal utterances to visual objects [77]. While they face many of the same challenges in dynamically constructing their networks, their task differs from ours in that they do not use their networks to perform object recognition. They use Bayesian networks to model the correspondence between the visual signal and the audio signal, but they constrain the "visual" recognition by supplying observations about the object's type and color. As a result, their network does not allow the speech signal to influence the visual interpretation of the scene. Our approach, on the other hand, uses a Bayesian network to model the structure of the shapes themselves, assuming that only low-level observations (e.g., the curvature of the stroke) can be observed. Although we have not yet tried it, we believe that speech and visual information could be integrated using our representation so as to allow the speech signal to influence the visual interpretation as well.

7.3 Summary

Online sketch recognition is a new and emerging field that has developed in response to the rise of pen-based technologies. We categorized the task of sketch recognition into three subtasks: beautification, gesture recognition and continuous sketch recognition. The work presented in this dissertation is among a small but growing body of work that attempts to solve most difficult of the three, continuous sketch recognition. To solve this problem, we draw from related work in speech and handwriting understanding, diagram understanding and computer vision. Although there is much work left to be done, this work presented in this dissertation fills an important gap in field of online sketch recognition: it presents a model that allows context to improve the recognition of continuously drawn, messy sketches.

Chapter 8

Conclusion

This dissertation has presented a new method of online continuous sketch recognition. Here we review its five contributions:

- We presented general framework for sketch recognition that can be extended to multiple domains. We described the framework and illustrated its use in two non-trivial domains.

- We presented a novel application of dynamically constructed Bayesian networks specifically developed for two-dimensional, constraint-based recognition. We illustrated through example that these successfully allow both context and stroke data to influence the system's interpretation of the sketch. We also demonstrated that these networks can be constructed automatically in response to the user's strokes.

- We showed that our system simultaneously segments and recognizes the objects in freely-drawn, often messy, sketches, relying on several methods for generating hypotheses for the user's strokes. This recognition process involved being able to reinterpret low-level interpretations based on higher-level context.

- We presented recognition results for the task of continuous sketch recognition in two domains: family trees and circuits. The sketches we collected will be made publicly available as an initial standard test set for this relatively unexplored task.

- Finally, based on the design and implementation of a SkRUI for Power Point diagram creation, we presented guidelines for building SkRUIs.

In summary, we have shown how to use context to improve online sketch interpretation and demonstrated its performance in SketchREAD, an implemented sketch recognition system that can be applied to multiple domains. We have shown that SketchREAD is more robust and powerful than previous systems at recognizing unconstrained sketch input in a domain. The capabilities of this system have applications both in human computer interaction and artificial intelligence. Using our system we will be able to further explore the nature of usable intelligent computer-based sketch systems and gain a better understanding of what people would like from a drawing system that is capable of understanding their freely-drawn sketches as more than just strokes. This work provides a necessary step in uniting artificial intelligence technology with novel interaction technology to make interacting with computers more like interacting with humans.

Appendix A

Shapes and Constraints

This Appendix lists all of the constraints and shape descriptions used in our experiments. Other shapes can be defined using any of the shapes here.

Table A.1: Primitive Shapes

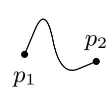
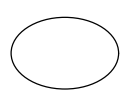
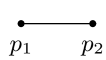
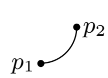
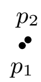
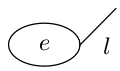
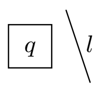
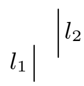
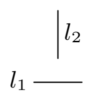
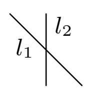
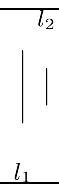
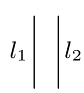
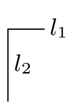
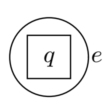
SHAPE	NAME	SUBCOMPONENTS
	Stroke	p_1, p_2
	Ellipse	
	Line	p_1, p_2
	Arc	p_1, p_2

Table A.2: Constraints

EXAMPLE	**NAME**	**ARGUMENTS**
	coincident	p_1, p_2
	touches	e, l
	nextTo	q, l
	parallel	l_1, l_2
	perpendicular	l_1, l_2
	intersects	l_1, l_2
	aligned	l_1, l_2
	equalLength	l_1, l_2
	smaller	l_1, l_2
	contains	e, q

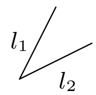
EXAMPLE	NAME	ARGUMENTS
	acuteAngle	l_1, l_2

Table A.3: Geometric Shapes

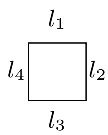
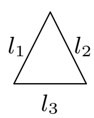
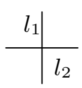
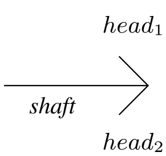
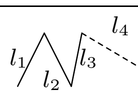
SHAPE	NAME	SUBSHAPES	CONSTRAINTS
	Quadrilateral	Line l_1 Line l_2 Line l_3 Line l_4	coincident $l_1.p_2$ $l_2.p_1$ coincident $l_2.p_2$ $l_3.p_1$ coincident $l_3.p_2$ $l_4.p_1$ coincident $l_4.p_2$ $l_1.p_1$
	Triangle	Line l_1 Line l_2 Line l_3	coincident $l_1.p_2$ $l_2.p_1$ coincident $l_2.p_2$ $l_3.p_1$ coincident $l_3.p_2$ $l_1.p_1$
	Plus	Line l_1 Line l_2	perpendicular l_1 l_2 intersects l_1 l_2
	Arrow	Line *shaft* Line $head_1$ Line $head_2$	coincident *shaft*.p_1 $head_1.p_1$ coincident *shaft*.p_1 $head_2.p_1$ acuteAngle $head_1$ *shaft* acuteAngle $head_2$ *shaft* equalLength $head_1$ $head_2$ larger *shaft* $head_1$
	Polyline	Vector Line $l[3, n]$	coincident $l[i].p_2$ $l[i+1].p_1$

Table A.4: Relationship Diagrams

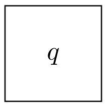
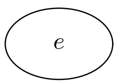
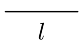
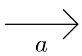
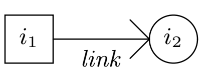
SHAPE	NAME	SUBSHAPES	CONSTRAINTS
Domain Shapes			
	Item-Quad	Quadrilateral q	
	Item-Ellipse	Ellipse e	
	Link-Line	Line l	
	Link-Arrow	Arrow a	
Domain Patterns			
	Connect-Line	Item i_1 Item i_2 Link-Line *link*	touches $link.l.p_1$ i_1 touches $link.l.p_2$ i_2
	Connect-Arrow	Item i_1 Item i_2 Link-Arrow $link$	touches $link.a.shaft.p_1$ i_1 touches $link.a.shaft.p_2$ i_2

Table A.5: Family Tree Diagrams

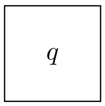
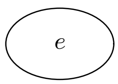
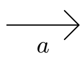
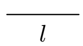
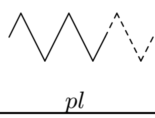
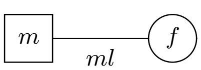
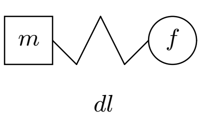
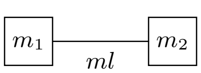
SHAPE	NAME	SUBSHAPES	CONSTRAINTS
Domain Shapes			
	Male	Quadrilateral q	
	Female	Ellipse e	
	Child-Link	Arrow a	
	Marriage-link	Link l	
	Divorce-link	Polyline pl	
Domain Patterns			
	Marriage	Male m Female f Marriage-link ml	touches $ml.l.p_1$ m touches $ml.l.p_2$ f
	Divorce	Male m Female f Divorce-link dl	touches $dl.pl.p_1$ m touches $dl.pl.p_2$ f
	Partner-Male	Male m_1 Male m_2 Marriage-Link ml	touches $link.l.p_1$ m_1 touches $link.l.p_2$ m_2

Table A.5: Family Tree Diagrams (*cont.*)

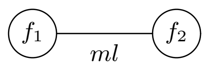
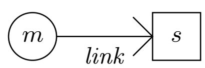
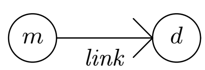
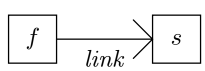
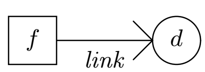
SHAPE	**NAME**	**SUBSHAPES**	**CONSTRAINTS**
	Partner-Female	Female f_1 Female f_2 Marriage-Link ml	touches $link.l.p_1$ f_1 touches $link.l.p_2$ f_2
	Mother-Son	Female m Male s Child-link $link$	touches $link.a.shaft.p_1$ m touches $link.a.shaft.p_2$ s
	Mother-Daughter	Female m Female d Child-link $link$	touches $link.a.shaft.p_1$ m touches $link.a.shaft.p_2$ d
	Father-Son	Male f Male s Child-link $link$	touches $link.a.shaft.p_1$ f touches $link.a.shaft.p_2$ s
	Father-Daughter	Male f Female d Child-link $link$	touches $link.a.shaft.p_1$ f touches $link.a.shaft.p_2.d$

Table A.6: Circuit Diagrams

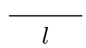
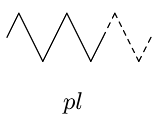
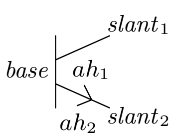
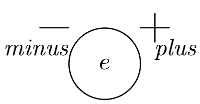
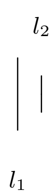
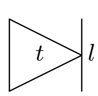
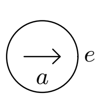
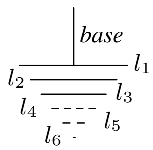
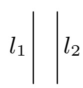
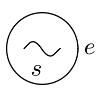
SHAPE	NAME	SUBSHAPES	CONSTRAINTS
Domain Shapes			
	Wire	Line l	
	Resistor	Polyline pl	acuteAngle $pl.l[i]$ $pl.l[i+1]$ equalLength $pl.l[i+1]$ $pl.l[i+2]$
	BJT	Line $base$ Line $slant_1$ Line $slant_2$ Line ah_1 Line ah_2	touches $slant_1.p_1$ $base$ touches $slant_2.p_1$ $base$ coincident $ah_1.p_1$ $ah_2.p_1$ touches $ah_1.p_1$ $slant_2$ touches $ah_2.p_1$ $slant_2$ equalLength ah_1 ah_2
	Voltage-Source	Ellipse e Line $minus$ Plus $plus$	nextTo $plus$ e nextTo $minus$ e
	Battery	Line l_1 Line l_2	smaller l_2 l_1 parallel l_1 l_2 nextTo l_1 l_2 aligned l_1 l_2
	Diode	Triangle t Line l	touches $t.l_2.p_2$ l equalLength $t.l_1$ l parallel $t.l_1$ l
	Current-Source	Ellipse e Arrow a	contains e a

Table A.6: Circuit Diagrams (*cont.*)

SHAPE	NAME	SUBSHAPES	CONSTRAINTS
	Ground	Line *base* Vector Line $l[3, 6]$	touches $base.p_1$ $l[1]$ perpendicular $base$ $l[1]$ parallel $l[i]$ $l[i+1]$ nextTo $l[i]$ $l[i+1]$ smaller $l[i+1]$ $l[i]$ aligned $l[i]$ $l[i+1]$
	Capacitor	Line l_1 Line l_2	parallel l_1 l_2 equalLength l_1 l_2 nextTo l_1 l_2 aligned l_1 l_2
	AC-Source	Ellipse e Stroke s	contains e s

Appendix B

Prior Probabilities, Primitives and Constraints

Here we list the values used in the Bayesian network and hypothesis generation algorithms. These values were determined empirically, in some cases through manual experimentation and in others through taking measurements from collected data.

B.1 Prior Probabilities

Table B.1 lists the prior probabilities we used for each of the top-level shapes and domain patterns in the family tree and circuit domains. We set these probabilities by hand based on our assessment of the relative probabilities of each shape or pattern. As discussed in Section 5.1, the system's performance was relatively insensitive to the exact values of these priors.

B.2 Properties and Constraints

Table B.4 lists each constraint, and gives the method by which properties measured from the applicable objects are translated into the corresponding *constraint measurement*. In addition, primitive shapes have related *shape measurements* that indicate how well the low-level data supports the primitive shape (Table B.3). Notation used in both tables is given in Table B.2.

The constraint measurements and shape measurements correspond to the measurement nodes linked to each primitive shape and constraint in the Bayesian network. The last column in Tables B.3 and B.4 gives $P(O_i|C_i)$ for each shape and constraint measurement, O_i, and its corresponding

Name	Prior Probability
Marriage	0.2
Divorce	0.01
Partnership (M and F)	0.001
Mother-daughter	0.2
Mother-son	0.2
Father-daughter	0.2
Father-son	0.2

(a) Family Trees

Name	Prior Probability
Wire	0.4
Resistor	0.05
Capacitor	0.05
Transistor	0.05
Ground	0.05
Diode	0.05
Voltage Source	0.05
Battery	0.05
A/C Source	0.05
Current Source	0.05

(b) Circuits

Table B.1: Prior probabilities for shapes and domain patterns in the family tree and circuit domains.

Notation	
o	Any Shape
s	A stroke or part of a stroke
l, a, e	Line, Arc, Ellipse
$l.p_1$	An endpoint of l
$l.m$	The midpoint of l
$I_{l_1 l_2}$	The intersection point of l_1 and l_2
b_o	Bounding box of shape o
$b_{o_1 o_2}$	Bounding box containing both o_1 and o_2
β_l	Perpendicular bisector of l
$o.c$	The center of the bounding box of shape o
l_s (a_s, e_s)	The best fit line (arc, ellipse) to stroke s
$\|\|s\|\|$	The length of stroke s
$lsqe(x, s)$	Least squared error between shape x and stroke s
$d(p_1, p_2)$	Distance between p_1 and p_2
$d(p_1, b_o)$	Distance between p_1 and the bounding box of o
θ_l	Orientation of l
$size(o)$	The size of a shape, defined for each shape.

Table B.2: Notation used in Tables B.3 and B.4.

Primitive Shapes				
Shape (S)	Shape Measurement (O)		CPT	
	Value (v)	Condition	$P(O = v\|S)$	$P(O = v\|\neg S)$
Line	0	$lsqe(l_s, s)/\|\|s\|\| < 0.5$	0.98	0.122
	1	$0.5 \leq lsqe(l_s, s)/\|\|s\|\| < 2.7$	0.01999999	0.258
	2	otherwise	0.0000001	0.620
Arc	0	$lsqe(a_s, s)/\|\|s\|\| < 0.167$	0.97	0.13
	1	$0.167 \leq lsqe(a_s, s)/\|\|s\|\| < 1.45$	0.015	0.248
	2	otherwise	0.005	0.622
Ellipse	0	$lsqe(e_s, s)/\|\|s\|\| < 0.1203$	0.8882	0.14
	1	$0.1203 \leq lsqe(e_s, s)/\|\|s\|\| < 1.29$	0.1006	0.136
	2	otherwise	0.0112	0.724

Table B.3: How shape measurements are derived from properties of the strokes.

constraint or primitive shape, C_i.

These CPTs were determined empirically from low-level data from users. Our goal was to estimate $P(O_i|C_i)$, or the probability distribution for some measurement of the user's stroke, O_i, given that the user was trying to (or trying not to) draw the primitive or constraint, C_i. Accordingly, we asked 27 users each to draw approximately 100 primitive shapes and constraints.

From this data, for each constraint we collected a set of instances in which the user intended to produce the shape or constraint, C_i and a set of instances in which the user intended to not produce C_i. For example, for the parallel constraint, the first set contained instances in which the user was told to draw parallel lines and the second set contained instances in which the user was told to draw acute angles, obtuse angles and perpendicular lines. We measured properties of the users strokes corresponding to C_i for each sketch in both sets. For example, for the constraint measurement pertaining to the constraint *parallel*, the property we used was the orientation difference between the two lines.

We then used these property values to determine simultaneously the best mapping from the continuous valued properties to the discrete valued measurements (O_i) as well as the distribution $P(O_i|C_i)$. We iteratively clustered the property values into an increasing number of clusters beginning with 3. A given iteration had n clusters, $G_i^0 ... G_i^n$, where the index of each cluster corresponds to the possible values for O_i. For each cluster, G_i^j, we measured the proportion of positive (and negative) examples that fell into that cluster, and used that value as $P(O_i = j|C_i)$ (and $P(O_i = j|\neg C_i)$. We stopped increasing the number of clusters when adding more clusters did not provide significant additional information in the conditional probability distribution. For all constraints, 3 seemed to

Constraints				
Constraint (C)	Constraint Measurement (O)		CPT	
	Value (v)	Condition	$P(O=v\|C)$	$P(O=v\|\neg C)$
Coincident	0	$d(l_1.p_1, l_2.p_1) < 20$	0.9487	0.1644
	1	$20 \leq d(l_1.p_1, l_2.p_1) < 30$	0.025	0.2637
	2	otherwise	0.0263	0.5719
Touches	0	$d(l_1.p_1, b_o) < 30$	0.7	0.2
	1	$30 \leq d(l_1.p_1, l_2.p_1) < 45$	0.2	0.3
	2	otherwise	0.1	0.5
NextTo	2	$d(c_{o_1}, c_{o_2}) > 75$ or $b_{o_1 o_2}$ contains $c_{o_i}, i \neq \{1,2\}$	0.0	1.0
	0	otherwise	1.0	0.0
Parallel	0	$\|\theta_{l_1} - \theta_{l_2}\| < 9$	0.8917	0.0923
	1	$9 \leq \|\theta_{l_1} - \theta_{l_2}\| < 18$	0.1083	0.0462
	2	otherwise	0.0	0.8615
AcuteAngle	0	$2 < \|\theta_{l_1} - \theta_{l_2}\| < 80$	0.8919	0.1111
	1	$\|\theta_{l_1} - \theta_{l_2}\| < 95$	0.0135	0.0635
	2	otherwise	0.0946	0.7302
Perpendicular	0	$84 < \|\theta_{l_1} - \theta_{l_2}\| < 96$	0.8917	0.0923
	1	$75 < \|\theta_{l_1} - \theta_{l_2}\| < 105$	0.1083	0.0462
	2	otherwise	0.0	0.8615
	0	β_{l_1} intersects l_2 and β_{l_2} intersects l_1	0.8917	0.0923
Intersects	0	$d(l_1.m, l_2) < 3/8 * \|\|l_1\|\|$ and $d(l_2.m, l_1) < 3/8 * \|\|l_2\|\|$	0.7	0.02
	1	$d(l_1.m, l_2) < 5/8 * \|\|l_1\|\|$ and $d(l_2.m, l_1) < 5/8 * \|\|l_2\|\|$	0.295	0.2
	2	otherwise	0.005	0.78
Aligned	0	β_{l_1} intersects l_2 and β_{l_2} intersects l_1	0.8917	0.0923
	1	β_{l_1} intersects l_2 or β_{l_2} intersects l_1	0.0923	0.0462
	2	otherwise	0.0	0.8615
Smaller	0	$size(o_1) < 0.8 * size(o_2)$	0.8917	0.0923
	1	$size(o_1) < size(o_2)$	0.1083	0.2462
	2	otherwise	0.0	0.6615
Contains	0	b_{o_1} contains b_{o_2}	1.0	0.0
	2	otherwise	0.0	1.0

Table B.4: How the constraint measurements are derived from properties of the strokes and shapes.

be a sufficient number of clusters and hence values for O_i.

Bibliography

[1] Bayesian network tools in java (bnj). http://bnj.sourceforge.net.

[2] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE pattern analysis and machine intelligence*, pages 522–525, 1993.

[3] Christine Alvarado and Randall Davis. Preserving the freedom of sketching to create a natural computer-based sketch tool. In *Human Computer Interaction International Proceedings*, 2001.

[4] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings. of IJCAI-2001*, August 2001.

[5] James Arvo and Kevin Novins. Fluid sketches: Continous recognition and morphing of simple hand-drawn shapes. In *UIST*, 2000.

[6] James Arvo and Kevin Novins. Smart text: A synthesis of recognition and morphing. In *AAAI Spring Symposium on Smart Graphics*, pages 140–147, Stanford, California, 2000.

[7] Elie Bienenstock, Stuart Geman, and Daniel Potter. Compositionality, mdl priors, and object recognition. In T. Petsche M. C. Mozer, M. I. Jordan, editor, *Advances in Neural Information Processing Systems 9*, pages 838–844. MIT Press, 1997.

[8] D. Blostein and L. Haken. Using diagram generation software to improve diagram recognition: A case study of music notation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11), 1999.

[9] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of UAI '96*, 1996.

[10] Michael C. Burl and Pietro Perona. Using hierarchical shape models to spot keywords in cursive handwriting. In *Proceedings of CVPR '98*, 1998.

[11] Anabela Caetano, Neri Goulart, Manuel Fonseca, and Joaquim Jorge. Sketching user interfaces with visual patterns. *Proceedings of the 1st Ibero-American Symposium in Computer Graphics (SIACG02)*, pages 271–279, 2002.

[12] Eugene Charniak. Bayesian networks without tears: making bayesian networks more accessible to the probabilistically unsophisticated. *Artificial Intelligence*, 12(4):50–63, 1991.

[13] Sung-Jung Cho and Jin H. Kim. Bayesian network modeling of hangul characters for on-line handwriting recognition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, 2003.

[14] P. R. Cohen, M. Johnston, D. R. McGee, S. L. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clowi. Quickset: Multimodal interaction for distributed applications. In *Proceedings of Mutimedia '97*, 1997.

[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. *3rd IAPR-TC15 Workshop on Graph-based Representation*, 2001.

[16] James M. Coughlan and Sabino J. Ferreira. Finding deformable shapes using loopy belief propagation. In *Proceedings of the Seventh European Conference on Computer Vision*, 2002.

[17] Christian Heide Damm, Klaus Marius Hansen, and Michael Thomsen. Tool support for cooperative object-oriented design: Gesture based modeling on an electronic whiteboard. In *CHI 2000*. CHI, April 2000.

[18] Fred DePiero and David Krout. Lerp: An algorithm using length-r paths to determine subgraph isomorphism. *Pattern Recognition Letters*, pages 33–46, 2003.

[19] Lee Erman, Frederick Hayes-Roth, Victor Lesser, and Raj Reddy. The hearsay-ii speech-understanding system: Integrating knowedge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[20] M. A. Fischler and R. A. Elschlager. The representation and matching or pictorial structures. *IEEE Transactions on Computers*, 22(1):67–92, January 1973.

[21] Robert B. Fisher. Polynomial-time geometric matching for object recognition. *International Journal of Computer Vision*, pages 37–61, 1997.

[22] Kenneth D. Forbus, Jeffry Usher, and Vernell Chapman. Sketching for military course of action diagrams. In *Proceedings of IUI 2003*, 2003.

[23] A. S. Forsberg, M. K. Dieterich, and R. C. Zeleznik. The music notepad. In *Proceedings of UIST '98*. ACM SIGGRAPH, 1998.

[24] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*, chapter 18. Prentice Hall, 2003.

[25] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[26] R. P. Futrelle and N. Nikolakis. Efficient analysis of complex diagrams using constraint-based parsing. In *ICDAR-95 (International Conference on Document Analysis and Recognition)*, pages 782–790, Montreal, Canada, 1995.

[27] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transations of Pattern Analysis and Machine Intelligence*, pages 721–741, 1984.

[28] Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[29] Sabine Glessner and Daphne Koller. Constructing flexible dynamic belief networks from first-order probabilistinc knowledge bases. In *Synbolic and Quantitatice Approaches to Reasoning and Uncertainty*, pages 217–226, 1995.

[30] Robert P. Goldman and Eugene Charniak. A language for construction of belief networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3), March 1993.

[31] W. Eric L. Grimson. The combinatorics of heuristic search termination for object recognition in cluttered environments. *IEEE Trans. on PAMI*, 13(9):920–935, September 1991.

[32] Mark Gross and Ellen Yi-Luen Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of UIST 96*, pages 183–192, 1996.

[33] Mark D. Gross. The electronic cocktail napkin - a computational environment for working with design diagrams. *Design Studies*, 17:53–69, 1996.

[34] Peter Haddawy. Generating bayesian networks from probability logic knowledge bases. In *Proceedings of UAI '94*, 1994.

[35] Tracy Hammond, 2005. PhD Thesis, Massachusetts Institute of Technology, to be published.

[36] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for uml class diagrams. *AAAI Spring Symposium on Sketch Understanding*, pages 59–68, March 25-27 2002.

[37] Tracy Hammond and Randall Davis. LADDER: A language to describe drawing, display, and editing in sketch recognition. *Proceedings of the 2003 Internaltional Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[38] Tracy Hammond and Randall Davis. Automatically transforming symbolic shape descriptions for use in sketch recognition. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[39] Richard Helm, Kim Marriott, and Martin Odersky. Building visual language parsers. In *Proceedings of CHI 1991*, pages 105–112, 1991.

[40] Jason Hong, James Landay, A. Chris Long, and Jennifer Mankoff. Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 73–77, March 25-27 2002.

[41] T. Igarashi, Satoshi Matsuoka, Sachiko Kawachiya, and Hidehiko Tanaka. Interactive beautification: A technique for rapid geometric design. In *UIST '97*, pages 105–114, 1997.

[42] D. L. Jenkins and R. R. Martin. Applying constraints to enforce users' intentions in free-hand 2-D sketches. *Intelligent Systems Engineering*, 1992.

[43] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[44] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Statistics for Engineering and Informaion Science. Springer, 2001.

[45] Rangachar Kasturi, Sing T. Bow, Wassim El-Masri, Jayesh Shah, James R. Gattiker, and Umesh B. Mokate. A system for interpretation of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):978–992, October 1990.

[46] Daphne Koller and Avi Pfeffer. Object-oriented bayesian networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty*, pages 302–313, Providence, RI, August 1-3 1997.

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

[48] James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of CHI 95*, pages 43–50, 1995.

[49] Edward Lank, Jeb S. Thorley, and Sean Jy-Shyang Chen. An interactive system for recognizing hand drawn UML diagrams. In *Proceedings for CASCON 2000*, 2000.

[50] Edward H. Lank. A retargetable framework for interactive diagram recognitiont. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR'03)*, 2003.

[51] Kathryn B. Laskey and Suzanne M. Mahoney. Network fragments: Representing knowledge for constructing probabilistic models. In *Proceedings of UAI '97)*, 1997.

[52] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[53] Jianming Liang, Finn V. Jensen, and Henrik I. Christensen. A framework for generic object recognition with bayesian networks. In *Proceedings of the First International Symposium on Soft Computing for Pattern Recognition*, 1996.

[54] A. Chris Long, Jr., James A. Landay, Lawrence A. Rowe, and Joseph Michiels. Visual similarities of pen gestures. In *Proceedings of the CHI 2000 conference on Human factors in computing systems*, 2000.

[55] Allan Christian Long. *Quill: a Gesture Design Tool for Pen-based User Interfaces*. Eecs department, computer science division, U.C. Berkeley, Berkeley, California, December 2001.

[56] Wei Lu, Wei Wu, and Masao Sakauchi. A drawing recognition system with rule acquisition ability. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 512–515, 1995.

[57] James V. Mahoney and Markus P. J. Fromherz. Handling ambiguity in constraint-based recognition of stick figure sketches. *SPIE Document Recognition and Retrieval IX Conf., San Jose, CA*, January 2002.

[58] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 105–112, Stanford, California, March 25-27 2002. AAAI Press.

[59] Kim Marriott and Bernd Meyer. Towards a hierarchy of visual languages. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, 1996.

[60] Kim Marriott, Bernd Meyer, and Kent Wittenburg. A survey of visual language specification and recognition. In Kim Marriott and Bernd Meyer, editors, *Visual Language Theory*, pages 5–85. Springer-Verlag, 1998.

[61] Nicolas Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, 1999.

[62] B. Messmer and H Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Trans. PAMI, Vol 20, 493 - 505*, 1998.

[63] Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay. DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18(3):259–324, 2003.

[64] Sharon Oviatt. Mutual disambiguation of recognition errors in a multimodal architecture. In *Proceeding of the CHI 99 Conference on Human Factors in Computing Systems: the CHI is the Limit*, pages 576–583, 1999.

[65] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[66] Avi Pfeffer, Daphne Koller, Brian Milch, and Ken Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Proceedings of UAI '99)*, pages 541–550, August 1999.

[67] David Poole. Probabilistic Horn abduction and Bayesnian networks. *Artificial Intelligence*, 1993.

[68] Dean Rubine. Specifying gestures by example. In *Computer Graphics*, volume 25(4), pages 329–337, 1991.

[69] Eric Saund, David Fleet, Daniel Larner, and James Mahoney. Perceptually supported image editing of text and graphics. In *Proceedings of UIST '03*, 2003.

[70] Tevfik Metin Sezgin. Phd thesis proposal. MIT PhD Thesis, to be published 2005.

[71] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. In *The Proceedings of 2001 Perceptive User Interfaces Workshop (PUI'01)*, Orlando, FL, November 2001.

[72] Michael Shilman, Hanna Pasula, Stuart Russell, and Richard Newton. Statistical visual language models for ink parsing. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 126–132, Stanford, California, March 25-27 2002. AAAI Press.

[73] Thomas Stahovich, Randy Davis, and Howrad Shrobe. Generating multiple new designs from a sketch. *Artificial Intelligence*, 104(1-2):211–264, 1998.

[74] Thomas M. Strat and Martin A. Fischler. Context-based vision: Recognizing objects using information from both 2-d and 3-d imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1050–1065, 1991.

[75] Antonio Torralba and Pawan Sinha. Statistical context priming for object detection. In *Proceedings of ICCV '01*, pages 763–770, 2001.

[76] Olya Veselova and Randall Davis. Perceptually based learning of shape descriptions. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

[77] S. Wachsmuth and G. Sagerer. Bayesian Networks for Speech and Image Integration. In *Proc. of AAAI-2002*, pages 300–306, Edmonton, Alberta, Canada, 2002.

[78] Luke Weisman. A foundation for intelligent multimodal drawing and sketching programs. Master's thesis, Massachusetts Institute of Technology, 1999.

[79] Yair Weiss. Belief propagation and revision in networks with loops. Technical report, Massachusetts Institute of Technology, November 1997. AI Memo No. 1616, CBCL Paper No. 155.

[80] Liu Wenyin, Wenjie Qian, Rong Xiao, and Microsoft Research China Xiangyu Jin. Smart sketchpad - an on-line graphics recognition system. In *Sixth International Conference on Document Analysis and Recognition (ICDAR '01)*, pages 1050–1054, 2001.

[81] M. E. Westling and L. S. Davis. Interpretation of complex scenes using Bayesian networks. *Lecture Notes in Computer Science*, 1352, 1997.

[82] Patrick H. Winston. Learning structural description from examples. *PVC*, 1975.

[83] Lizhong Wu, Sharon L. Oviatt, and Philip R. Cohen. Multimodal integration—a statistical view. *IEEE Transaction on Multimedia*, 1(4):334–341, December 1999.