Online Sketch Recognition from a Dynamic Perspective

by

Tevfik Metin Sezgin

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

May 2006

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2006

Certified by .....
Randall Davis
Professor
Thesis Supervisor

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

Online Sketch Recognition from a Dynamic Perspective

by

Tevfik Metin Sezgin

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Sketching is a natural mode of interaction used in a variety of settings. For example, people sketch during early design and brainstorming sessions to guide the thought process; when we communicate certain ideas, we use sketching as an additional modality to convey ideas that can not be put in words. The emergence of hardware such as PDAs and Tablet PCs has enabled capturing freehand sketches, enabling the routine use of sketching as an additional human-computer interaction modality.

But despite the availability of pen based information capture hardware, relatively little effort has been put into developing software capable of understanding and reasoning about sketches. To date, most approaches to sketch recognition have treated sketches as images (i.e., static finished products) and have applied vision algorithms for recognition. However, unlike images, sketches are produced incrementally and interactively, one stroke at a time and their processing should take advantage of this.

This thesis explores ways of doing sketch recognition by extracting as much information as possible from temporal patterns that appear during sketching. We present a sketch recognition framework based on hierarchical statistical models of temporal patterns. We show that in certain domains, stroke orderings used in the course of drawing individual objects contain temporal patterns that can aid recognition. We build on this work to show how sketch recognition systems can use knowledge of both common stroke orderings and common object orderings. We describe a statistical framework based on Dynamic Bayesian Networks that can learn temporal models of object-level and stroke-level patterns for recognition. Our framework supports multi-object strokes, multi-stroke objects, and allows interspersed drawing of objects – relaxing the assumption that objects are drawn one at a time. Our system also supports real-valued feature representations using a numerically stable recognition algorithm. We present recognition results for hand-drawn electronic circuit diagrams. The results show that modeling temporal patterns at multiple scales provides a significant increase in correct recognition rates, with no added computational penalties.

Thesis Supervisor: Randall Davis
Title: Professor

Contents

1 Introduction **12**

1.1 Why do sketch recognition? 12

1.2 What is sketch recognition? 15

1.3 Sketching is more than Graffiti 16

1.4 Combinatorics makes sketch recognition hard 17

1.5 Using temporal patterns allows efficient sketch recognition 18

1.6 Thesis roadmap 21

2 Properties of sketches **22**

2.1 Static properties of sketches 22

2.2 Dynamic properties of sketches 23

2.3 Temporal patterns appear to be common in sketching 26

3 Approach **29**

3.1 The intuition 29

3.2 Desired features of a model 30

3.2.1 Handling stroke-level patterns 30

3.2.2 Support for multiple classes and drawing orders 30

3.2.3 Handling variations in encoding length 31

3.2.4 Probabilistic matching score 31

3.2.5 Learning compact representations 31

3.2.6 Rich feature representation 31

3.2.7 Handling object level patterns 32

3.2.8 Ability to handle interspersed drawing 32

3.3 Terminology and problem formulation 33

3.3.1 Terminology . . . 33
3.3.2 Problem formulation . . . 34
3.4 Choice of probabilistic model and input representation . . . 36
3.4.1 Hidden Markov Models . . . 36
3.4.2 Dynamic Bayesian Networks . . . 37
3.4.3 The input . . . 37
3.5 Flat model . . . 37
3.5.1 Encoding . . . 38
3.5.2 Modeling with fixed input length HMMs . . . 38
3.5.3 Modeling using HMMs with variable length training data . . . 40
3.5.4 Advantages of the graph based approach . . . 41
3.6 Multiscale model . . . 42
3.6.1 Encoding . . . 42
3.6.2 Model description . . . 43
3.6.3 Implementation issues . . . 45
3.7 Multiscale-interspersing model . . . 46
3.7.1 Switching parents . . . 48
3.7.2 Description of the model topology . . . 49
3.8 Discussion . . . 52

4 Results **53**

4.1 Circuit data collection . . . 53
4.2 Generating an observation sequence . . . 54
4.3 Modeling object level patterns increases recognition accuracy . . . 55
4.3.1 Experiment setup and quantitative results . . . 55
4.3.2 Qualitative examples and discussion . . . 57
4.4 Handling interspersings increases recognition accuracy . . . 57
4.4.1 Experiment setup and quantitative results . . . 57
4.4.2 Quantifying errors per-interspersing . . . 60
4.4.3 Qualitative examples and discussion . . . 61
4.5 Summary . . . 65

5 Related work **66**

5.1 Sketch based interfaces 66

5.2 Recognition systems 66

5.2.1 Graphics recognition 66

5.2.2 Handwriting recognition 66

5.2.3 Symbol recogntion systems 66

5.3 Sketch recognition systems 66

5.3.1 Use of spatial data for recognition 67

5.3.2 Use of temporal data for recognition 67

6 Contributions **68**

6.1 Contributions 68

6.2 Discussion 69

7 Future work **71**

7.1 Incorporating spatial features 71

7.2 Exploiting multi-scale feature representations 71

7.3 Handling handwriting 71

List of Figures

1-1 Examples of applications that use and manipulate graphical representations of objects. We believe such applications will benefit most from sketch based interfaces. 13

1-2 In most design software, toolbars serve as the primary means of entering graphical information. Here we see toolbars that are part of the user interface for the applications shown in Fig. 1-2. 14

1-3 The goal of sketch recognition is to enable people to convey graphical information by drawing. Imagine if an electrical engineer could draw (a) and have the computer automatically construct the PSpice model in (b). 14

1-4 A circuit digram illustrating what we mean by a sketch. Our goal is to group strokes forming the same object together (segmentation) and determine what object they form (classification). In this case segmentation finds the circled grouping and classification indicates that they represent an *NPN transistor*. 15

1-5 Gesture based Graffiti input scheme for PDAs. Input gestures must be drawn in a single stroke, starting at the heavy dot. Shapes of the some gestures do not look like the intended symbols (e.g., '%', '@'). . 17

1-6 Circuit diagram for an amplifier. Stroke ordering for a fragment of the circuit is indicated by numbers. 19

1-7 In this example, the user interspersed a wire connected to the collector of the transistor **Q2**. We present a model that can recognize objects correctly even in the presence of such interspersings and identify interspersed objects (indicated by coloring the interspersed wire in black instead of cyan). 19

2-1 Example showing what we mean by a sketch. Note the messy, freehand nature of the diagram. 23
2-2 Police sketches (left) and artistic renderings. Recognition of such sketches fall outside the scope of this thesis. 24
2-3 Results of the "grape clusters" experiment by van Sommers [81]. Users were asked to copy each cluster on paper. Numbers in each region show the average order in which the boundary was drawn and shows the strong effects of planning and anchoring. 25
2-4 Examples of the figures that the users were asked to draw in the user study. 27
2-5 A sketching style diagram showing two ways of drawing stick-figures . 28

3-1 Symbols for *stop* and *skip-audio-track* (on the left), and a sketched *stop* symbol (right). 30
3-2 An example showing a multi-object stroke. The selected circuit fragment contains two wires and a resistor drawn using a single stroke. . 34
3-3 The dynamic Bayesian network representing the model for capturing stroke level patterns. This fragment has a dual representation as a hidden Markov model with continuous observations and **M** mixtures. 43
3-4 The dynamic Bayesian network representing the multiscale model. . . 44
3-5 An example of interspersing: The user draws two other objects (wires #3 and #6) over the course of drawing the transistor. The drawing order is indicated by numbers. 47
3-6 The dynamic Bayesian network representing our model that uses multiscale patterns and handles interspersed drawing. 48
3-7 Example illustrating the switching parent mechanism. 49

4-1 Examples of collected circuits. 54
4-2 Conversion to the primitive sequence. 55
4-3 One of the circuits drawn by our study participants. Stroke ordering for a fragment of the circuit is indicated by numbers. 58
4-4 Examples of errors corrected using context provided by object-level patterns. There are four misrecognitions in (a). Note how two of these misrecognitions is fixed using knowledge of object level patterns (resistor in the upper left corner (**R1**) and the wire connected to the emitter of the misrecognized transistor). 59
4-5 The dynamic Bayesian network representing the multiscale-interspersing model that handles interspersed drawing. 60
4-6 Another circuit drawn by our study participants. Stroke ordering for a fragment of the circuit is indicated by numbers. 62
4-7 In this example, the user interspersed a wire connected to the collector of the transistor on the right. As a result the transistor is misclassified by the multiscale model (a). The multiscale-interspersing model identifies the wire interspersing (indicated by black) and correctly interprets the transistor (b). 63
4-8 Another example of handling interspersing. In this case the errors caused by the interspersing is minor compared to 4-7. The interspersed wire is interpreted to be part of the transistor. The multiscale-interspersing model identifies the wire interspersing (indicated by black in b). 64

6-1 An example of strokes with the right temporal character but wrong shape being classified incorrectly (reproduced from Chap. 4). The two vertical lines (l_1, l_2) are offset from one another, but were classified to be a capacitor because they were drawn one after the other, and the temporal features extracted from them agree with the capacitor model that we learned. 70

List of Tables

2.1 A summary of the statistics from our user study for a subset of the symbols drawn by the users (shown in Fig.2-4). Note that 120=5!, 720=6! and 40320=8!. Users drew 30 or more examples of each object. 27

4.1 Mean correct recognition rates for the baseline system (μ_b) and the multiscale model (μ_m) on sketches collected from 8 participants. The table also shows the percentage reductions in the error rates and maximum error reductions achieved for each user as percentages (Δ_{err} and $max\Delta$). On average, modeling object-level patterns always improves performance. 56

4.2 Mean correct recognition rates for the multiscale-interspersing (μ_{m-i}) and the multiscale models (μ_m) for users who exhibited interspersed drawing behavior. The table also shows the percentage reductions in the error rates and maximum error reductions achieved for each user as percentages (Δ_{err} and $max\Delta$). On average, handling interspersing patterns always improves performance. 58

4.3 This table shows the number of misrecognized primitives per interspersing. The first two row shows the number of total primitives in each sketch and the number of added interspersings. Next two rows show the number of primitives incorrectly recognized by the baseline and by our model. The next row shows the number of primitives that the baseline misses because it cannot handle interspersings and the last row is the number of primitives missed by the baseline per interspersing. 61

Chapter 1

Introduction

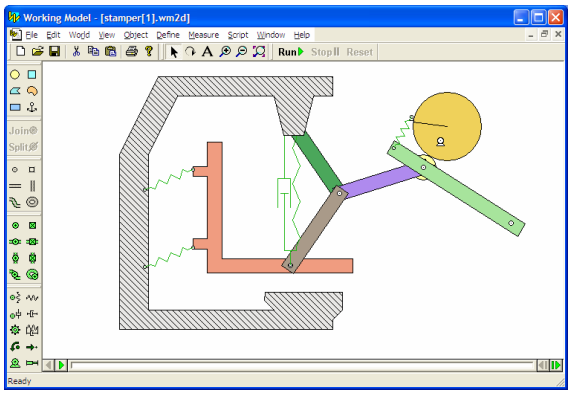
1.1 Why do sketch recognition?

Sketching is a natural modality of communication employed in a wide variety of settings. People sketch during early design and brainstorming sessions to guide the thought process and use sketches as a means of documentation[80]. We use sketching to convey thoughts that can not be put in words.

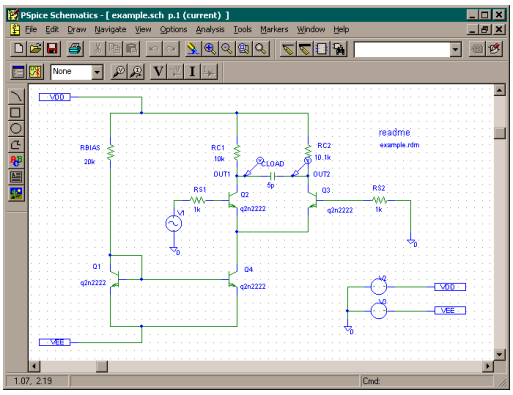
The increasing availability of pen based hardware such as PDAs and Tablet PCs makes capturing sketches easier than ever. Despite their ubiquity, suitability for certain HCI tasks, and the availability of supporting hardware, there is little computer support for sketching.

We believe making computers "sketch literate" will produce smarter, more natural user interfaces for a host of computer applications. Applications to benefit most from sketch based interfaces are the ones that use and manipulate graphical representations of objects. Fig. 1-1 shows examples of such commercial applications: a 2D rigid body physics simulator (Fig. 1-1.a), an analog circuit simulator (Fig. 1-1.b), a UML design tool (Fig. 1-1.c), and PowerPoint (Fig. 1-1.d).

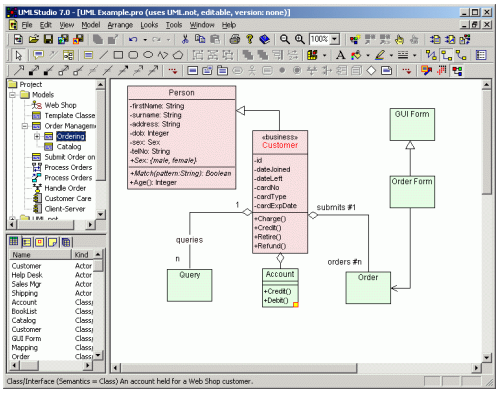
All of these applications currently share the same user interface paradigm based on mouse and keyboard input. Toolbars of the sort shown in Fig. 1-2 serve as the primary means of entering graphical information. In most cases, changing the default parameters of the graphical objects requires further keyboard and mouse input through menus and dialog boxes.



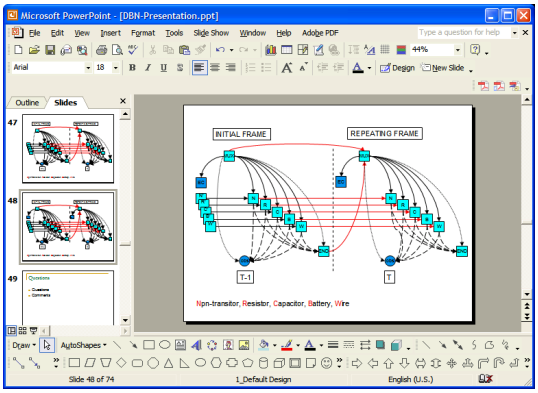
(a) Working Model 2D



(b) PSpice



(c) UMLStudio



(d) PowerPoint

Figure 1-1: Examples of applications that use and manipulate graphical representations of objects. We believe such applications will benefit most from sketch based interfaces.

Our goal in building sketch recognition systems is to allow people to convey graphical information in the same way that they have done for thousands of years: simply by drawing. Imagine, for example, if an electrical engineer could draw the freehand circuit diagram in Fig. 1-3.a and have the computer automatically construct the PSpice model in Fig. 1-3.b. This is what we mean by sketch recognition and in this thesis, we show how it can be done by extracting as much information as it is practically possible from temporal patterns that appear during sketching.

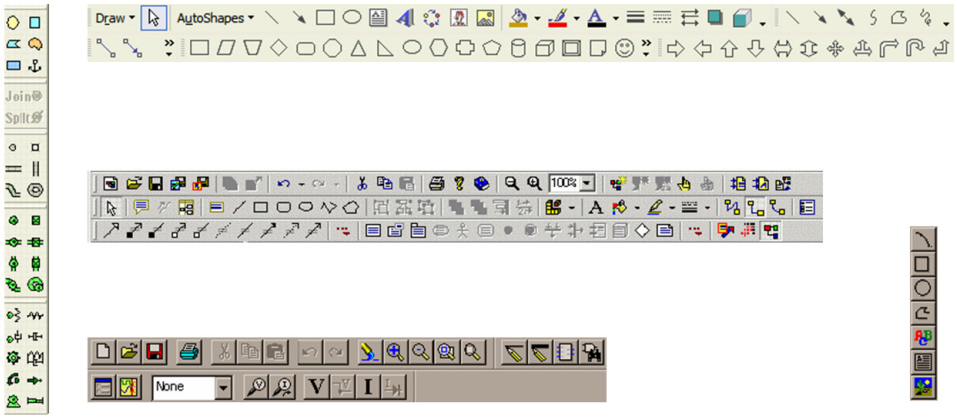
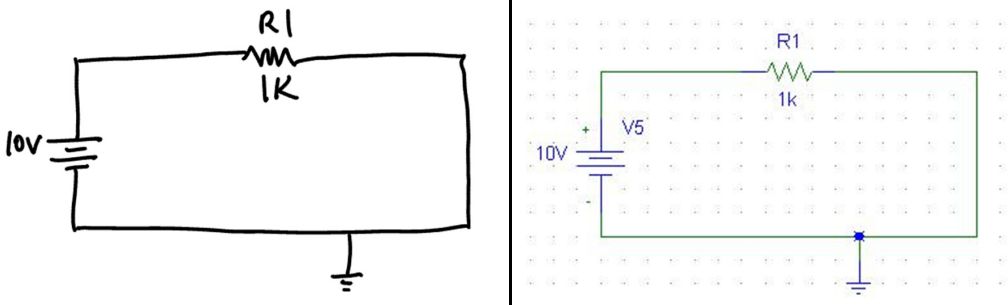


Figure 1-2: In most design software, toolbars serve as the primary means of entering graphical information. Here we see toolbars that are part of the user interface for the applications shown in Fig. 1-2.



(a) A freehand circuit diagram.

(b) What we wish to obtain from (a).

Figure 1-3: The goal of sketch recognition is to enable people to convey graphical information by drawing. Imagine if an electrical engineer could draw (a) and have the computer automatically construct the PSpice model in (b).

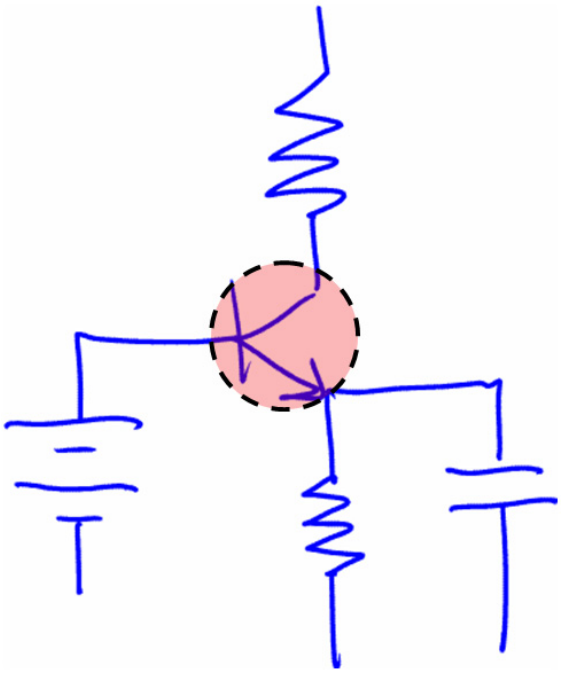


Figure 1-4: A circuit digram illustrating what we mean by a sketch. Our goal is to group strokes forming the same object together (segmentation) and determine what object they form (classification). In this case segmentation finds the circled grouping and classification indicates that they represent an *NPN transistor*.

1.2 What is sketch recognition?

The basic task of *sketch recognition* can be understood best by considering the input and the output.

By a *sketch*, we mean messy, informal hand-done drawings (e.g., Fig. 1-4). Specifically we are interested in recognizing sketches with objects of a symbolic nature that can be represented using structural descriptions, which have been the focus of the sketch recognition community [4, 73, 55, 29].

It is worth emphasizing that we are interested in sketches that are drawn naturally as the user would draw on a piece of paper. This is unlike some systems that require users to draw each object using a single stroke, or pause after drawing each object to facilitate segmentation, thereby reducing the problem of sketch recognition to that of isolated symbol recognition.

We specifically refrain from forcing the user to sketch in a certain way or follow certain conventions during sketching. Consequently, our recognition algorithms will

have to deal with unsegmented sketches with multi-stroke objects and drawing with interspersed objects (i.e., starting a new object before completing the current one).

Earlier we gave an informal definition of sketch recognition as the process of automatically converting freehand input into a representation which can be used by domain specific applications. A more formal definition of sketch recognition can be characterized in terms of three tasks:

- ***Segmentation:*** The task of grouping strokes so that those constituting the same object end up in the same group. At this point it is not known what object the strokes form. For example, in Fig. 1-4, the correct segmentation gives us fifteen disjoint sets of strokes (assuming each wire segment is a single object).

- ***Classification:*** Classification is the task of determining which object each group of strokes represents. For Fig. 1-4, recognition would indicate that the circled strokes represent an *NPN transistor*.

- ***Labeling:*** Labeling is the task of assigning labels to components of a recognized object (e.g., identifying the terminal with the current symbol in the NPN transistor in Fig. 1-4 as the emitter terminal).

Although the three tasks described above are conceptually separate, in any real sketch recognition scenario they may be intermixed. For example, it could be the case that a particular recognition architecture classifies objects and labels their constituent parts simultaneously. More likely, segmentation is done first, followed by classification and labeling.

1.3 Sketching is more than Graffiti

Graffiti is a single stroke gesture recognition scheme developed mainly for PDAs. One of the most popular gesture recognition methods for Graffiti-like input is described by Rubine [68]. Fig. 1-5 shows the Graffiti alphabet for punctuation and various symbols.

True sketch recognition is a different and much more challenging problem than recognizing the carefully prescribed strokes used in Graffiti. Unlike freehand drawing,

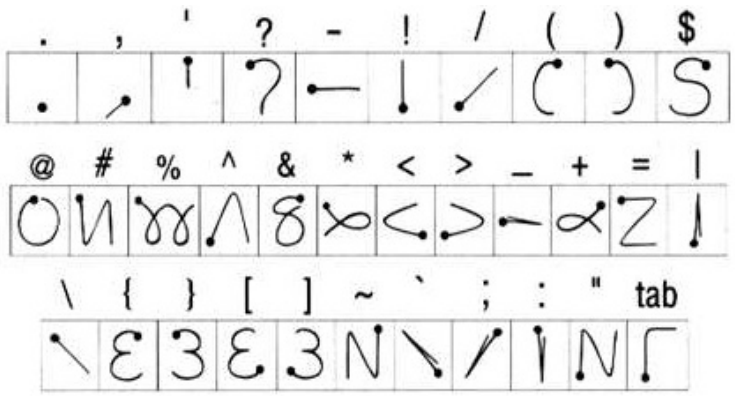


Figure 1-5: Gesture based Graffiti input scheme for PDAs. Input gestures must be drawn in a single stroke, starting at the heavy dot. Shapes of the some gestures do not look like the intended symbols (e.g., '%', '@').

the Graffiti alphabet comes with a number of conventions and restrictions that make it easy to recognize but unnatural. In particular:

- Symbols are specified by gestures that do not necessarily resemble the target shapes (e.g. '%', '@' in Fig. 1-5).
- Input gestures must be drawn in a single stroke.
- The gesture must start at a predetermined point (the heavy dots shown in Fig. 1-5).
- The starting point may affect classification because some symbols are specified with the same gesture but different starting points (e.g., [and { in Fig. 1-5).

These restrictions conflict with our goal of supporting freehand drawing where users can sketch without worrying about adhering to certain drawing guidelines.

1.4 Combinatorics makes sketch recognition hard

Treating sketches as images leads to recognition algorithms with exponential time complexities [55]: subgraph isomorphism-based methods and correspondence-based approaches have exponential time complexities. If we assume that we have m object classes and that each object model has k components, a simple calculation shows that

in the worst case, recognizing an object surface with n strokes requires $\binom{n}{k}$ grouping operations and $k!$ constraint checking operations, yielding a total of $m\binom{n}{k}k!$ operations. In practice, the combinatorics get even worse because sketches are inherently noisy (e.g. due to digitization) and messy (e.g. sloppy drawing by the user).

Exponential time and space requirements are unacceptable for interactive sketch recognition, because giving users feedback about the recognition results is an essential part of sketch based interfaces[4]. Sketch recognition algorithms proposed so far either don't run in real-time [4], or make limiting assumptions about the domains (e.g., maximum object size, maximum number of strokes per object [75, 54, 53]). Furthermore most of these approaches have been demonstrated only in domains where objects are non-touching and hence segmentation is less of an issue compared to more challenging domains where objects may overlap or touch one another (e.g., box-connector type diagrams such as UML diagrams, family trees, circuit diagrams).

1.5 Using temporal patterns allows efficient sketch recognition

This thesis describes an approach to efficient sketch recognition. We show how treating sketching as an incremental and interactive process enables polynomial time recognition algorithms, by allowing us to exploit naturally appearing temporal patterns in sketching.

Current sketching systems are indifferent to who is using the system, employing the same recognition routines for all users. But user studies we have conducted indicate clearly that different users have different sketching preferences and styles, and that there is considerable value in being able to capture these different styles.

A major component of individual drawing styles is the temporal patterns seen during sketching. For example, when people draw stick figures, one frequently seen stroke-level pattern is a sequence consisting of a circular stroke, followed by a vertical line, followed by two pairs of positively and negatively sloped lines – respectively corresponding to the head, body, arms and legs of the stick-figure. It is these kinds of temporal patterns that can aid sketch recognition. Using temporal patterns allows

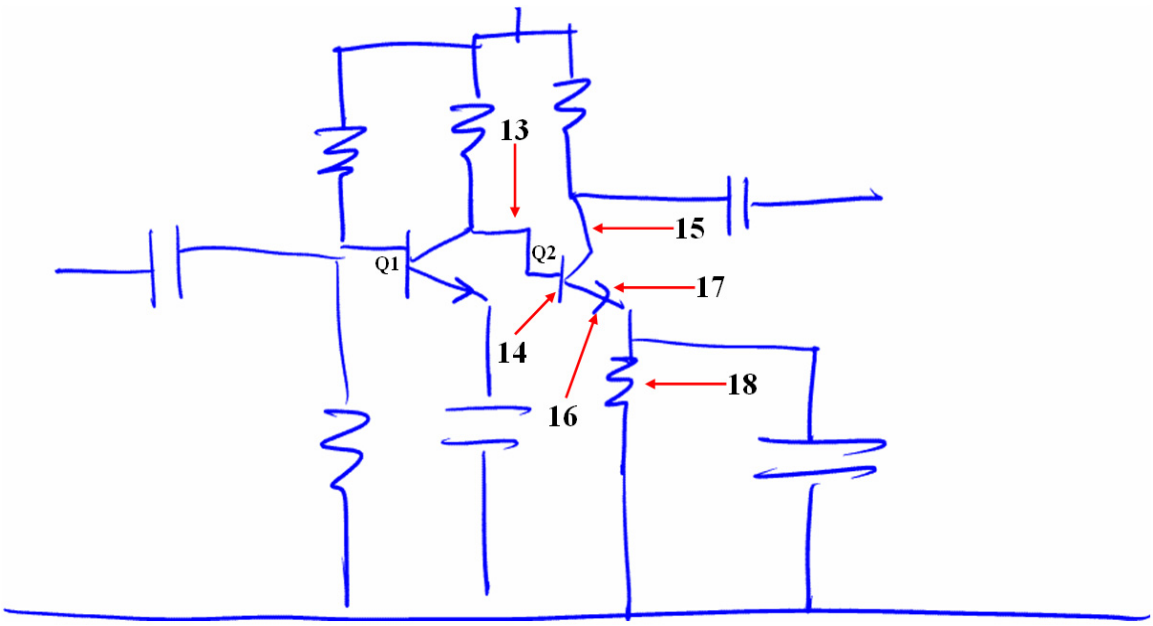


Figure 1-6: Circuit diagram for an amplifier. Stroke ordering for a fragment of the circuit is indicated by numbers.

us to work with one dimensional time series data, for which there are efficient, sound mathematical analysis methods.

Consider the circuit diagram in Fig. 1-6, showing an amplifier circuit and the stroke ordering for a fragment of the circuit of particular interest to us. Using the hierarchical probabilistic models that we present in this thesis, we can learn multiscale temporal patterns that naturally exist in the creation of such a sketch and interpret it under a second on a standard PC. Our hierarchical models capture knowledge of both common stroke orderings and common object orderings. Results of our evaluation involving circuit diagrams collected from eight participants show that modeling common object orderings reduces the number of recognition errors by 14%- 37%.

Furthermore our temporal model explicitly models interspersed drawing behavior and does not require the users to draw one object at a time. This means we can recognize sketches correctly even if the user starts drawing a new object before finishing the current one. For example, we can correctly recognize the circuit in Fig. 1-6 even though a wire (part of stroke #15) was drawn in the course of drawing the transistor **Q2**. Fig. 1-7 shows the interpretation of our system. Such interspersed drawing behavior poses serious challenges to a naïve temporal model for sketch recognition,

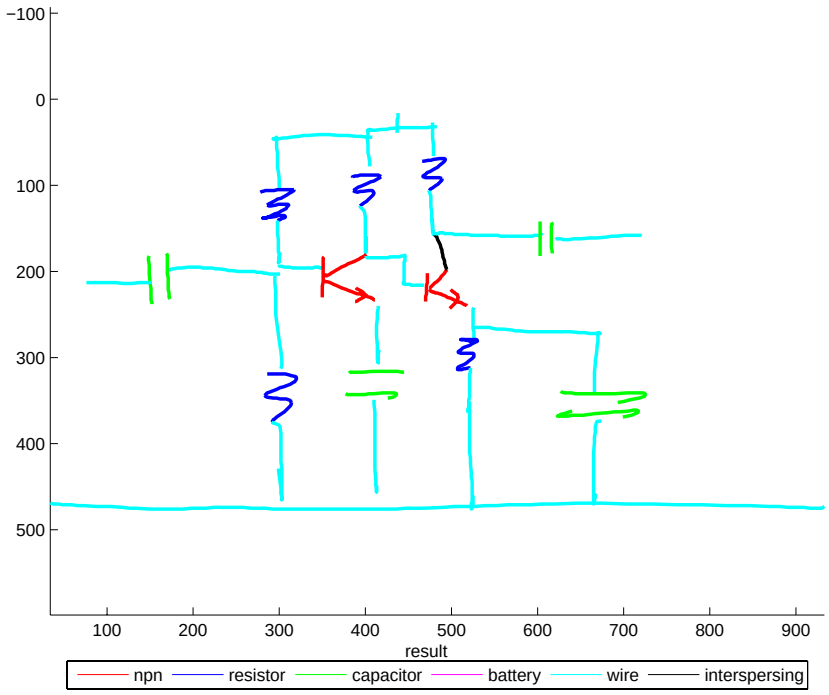


Figure 1-7: In this example, the user interspersed a wire connected to the collector of the transistor **Q2**. We present a model that can recognize objects correctly even in the presence of such interspersings and identify interspersed objects (indicated by coloring the interspersed wire in black instead of cyan).

but we show how the issue can be addressed remaining within the formal framework of Dynamic Bayesian Networks. We show that modeling interspersing reduces recognition errors by an additional 20%-37%.

Finally our recognition framework supports multi-object strokes (e.g., stroke #15 in Fig. 1-6 represents a wire and part of the transistor **Q2**), multi-stroke objects (e.g., **Q1** consists of four strokes), objects with a variable number of components (e.g., some resistors have fewer humps). We also support discrete and real-valued feature representations which allows us to encode our data using discrete encodings for categorical properties (e.g., a stroke being a circle vs. a line), and real-valued encodings for continuous features (e.g., length, orientation of a line). We report that a numerically stable belief propagation algorithm known as the Lauritzen-Jensen stable conditional Gaussian belief propagation algorithm should be used for modeling discrete and real-valued features.

1.6 Thesis roadmap

The next chapter summarizes properties of sketches and reports results from a user study that we conducted to measure the degree to which people use predictable stroke orderings when they sketch. Chapter three gives a formal description of the sketch recognition problem and introduces our model that uses hierarchical graphical models to model multiscale patterns in sketching while supporting interspersed drawing. Chapter three also introduces two baseline methods that we use in our evaluation. In chapter four, we report evaluation results comparing the performance of our model to the two baseline models. Chapters five and six summarize the related work and our main contributions. We conclude with future work.

Chapter 2

Properties of sketches

Here we briefly summarize properties of sketches because it is important to understand the nature of the data that we are dealing with.

2.1 Static properties of sketches

The static properties of sketches are those that would be present even if all we had was a scanned image of the sketch. They include image-like properties such as digitization noise and others that are specific to sketches, such as messiness.

Noise arises from digitization performed by a digitizing tablet, a scanner, or another imaging device. Each device has an inherent resolution and noise characteristics.

Messiness refers to phenomena such as overshooting, undershooting, messiness in the lines and jitters due to hand tremor. For example, in Fig. 2-1, stroke segments making the humps of the resistors don't always make the same angle, and some humps are relatively shorter although in an ideal resistor symbol, each segment is perfectly linear and they make the same angle. The missing connection between the capacitor in Fig. 2-1 and the wire above it is an example of undershooting – another kind of messiness. It is these kinds of phenomena resulting from the kinesthetic nature of sketching that makes sketch recognition a much harder problem than diagram recognition studied extensively by the document analysis community [77, 9].

Sketches are highly informal. Instances of the same icon may have varying aspect ratios and scales. Parts (subcomponents) of an object may have varying aspect ratios

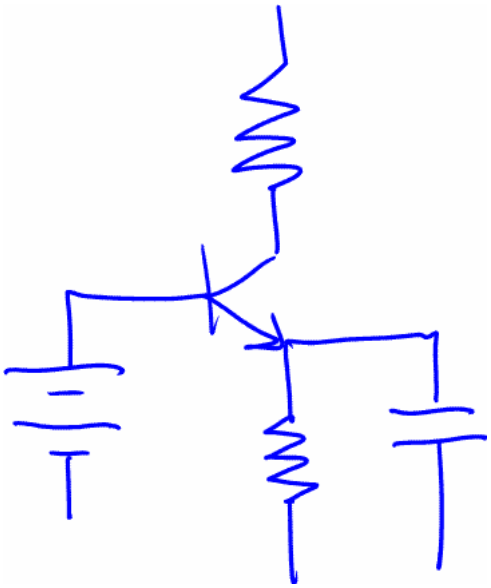


Figure 2-1: Example showing what we mean by a sketch. Note the messy, freehand nature of the diagram.

within the object. This high variability rules out popular transformation space search approaches that rely on affine properties of images.

The sketches we deal with are mostly iconic (e.g., a "light bulb" representing an idea, or a stick-figure representing a person). Their iconic nature makes it possible to have symbolic descriptions of sketches. There are also domains with highly non-iconic properties, such as "police sketches" or artistic renderings which fall outside the scope of this thesis (Fig. 2-2).

Most sketches can be broken down into compositions of simpler objects, and those objects can be further broken down into primitives such as lines, circles etc. For example, a simple sketch of a car can be broken down into the body, and the wheels; and the wheels can be further broken down into two circles representing the wheel and the axle.

2.2 Dynamic properties of sketches

One of the advantages of online sketching is that a sketch can be captured as it is constructed, making it possible to capture properties that may potentially aid recognition. Our stroke-based representation allows us to access stroke level information

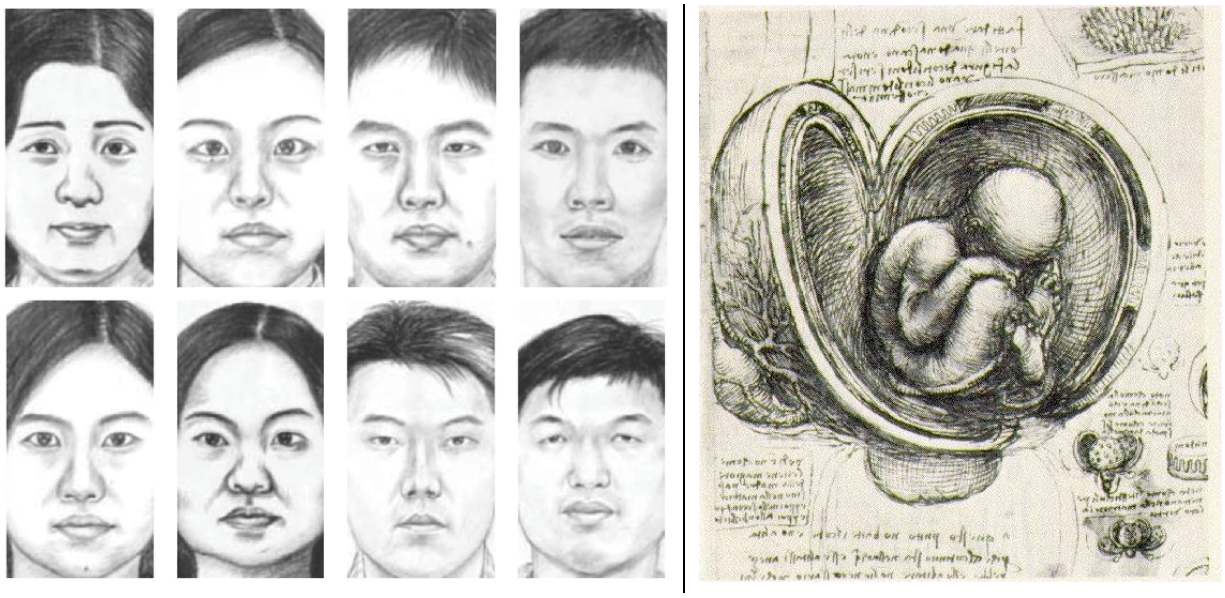


Figure 2-2: Police sketches (left) and artistic renderings. Recognition of such sketches fall outside the scope of this thesis.

including what order the strokes came in, as well as timing information for individual points.

Because sketching is incremental (i.e, strokes are put on the sketching surface one at a time) the sketching surface will at almost every moment have on it a partially drawn object. Partially drawn objects are a common source of ambiguity in sketches. They can act as spurious strokes and confuse recognition algorithms. Partially drawn objects also raise the difficult issue of balancing the desire to generate all plausible recognition hypotheses and the desire to wait until there are enough components so the search is more constrained. The methods that we present explicitly address these issues by modeling the sketching process using a set of variables that, among other things, capture our belief about whether the user has done drawing the current object.

In a typical sketch recognition scenario, there is two way communication: information flows from the user to the computer via the strokes drawn and the editing operations, and from the computer to the user via computer's display of its interpretation of the strokes and editing operations. This interactive nature of sketching is an important source of knowledge when it comes to confirming the correctness of a system's interpretation. For example, the longer the user lets an interpretation exist,

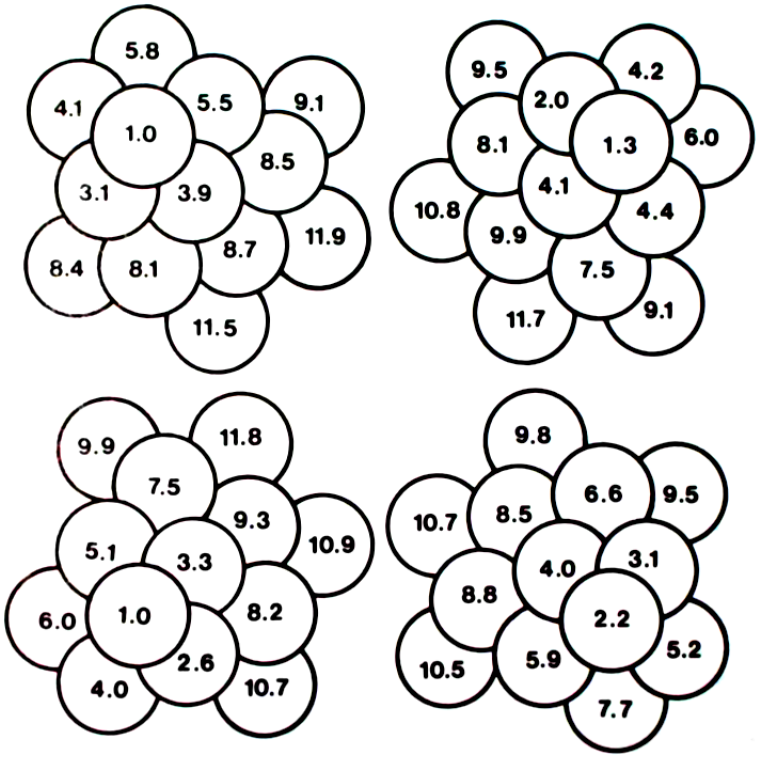


Figure 2-3: Results of the "grape clusters" experiment by van Sommers [81]. Users were asked to copy each cluster on paper. Numbers in each region show the average order in which the boundary was drawn and shows the strong effects of planning and anchoring.

the more certain the system can be about its interpretation [6]. It is therefore important to have recognition algorithms that are fast enough to provide realtime feedback to the user.

Sketching is highly stylized in the sense that people have strong biases in the way they sketch (e.g., people draw enclosing objects first, use a left-to-right stroke ordering when drawing symmetric objects). There is psychological evidence attributing such phenomena to motor convenience, part saliency, hierarchy, geometric constraints, planning and anchoring [78, 81]. For example, Fig. 2-3 shows the results of the "grape clusters" experiment described by van Sommers [81] where the users were asked to copy four clusters on paper. Numbers in each region show the average order in which the boundary was drawn and shows the strong effects of planning and anchoring.

Existence of ordering patterns during drawing is significant from a recognition

perspective because the regularities in sketching can be used for recognition. Because this forms the basis for our approach to recognition presented chapter 3, and we conducted user studies to validate this observation for domains of interest to us.

2.3 Temporal patterns appear to be common in sketching

We ran a user study to assess the degree to which people have sketching styles, i.e., a reasonably consistent stoke order used when drawing an item. For example, if one starts drawing a stick-figure with the head, then draws the torso, the legs and the arms respectively, we regard this as a style different from the one where the arms are drawn before the legs (see Fig. 2-5).

Our user study asked users to sketch various icons, diagrams and scenes from six domains. Example tasks included drawing:

- Finite state machines performing simple tasks such as recognizing a regular language.
- Unified Modeling Language (UML) diagrams depicting the design of simple object-oriented programs.
- Scenes with stick-figures playing certain sports.
- Course of Action Diagram symbols used in the military to mark maps and plans.
- Digital circuit diagrams that implement a simple logic expression.
- Emoticons expressing happy, sad, surprised and angry faces.

We asked 10 subjects to sketch three sketches from each of the six domains, collecting a total of 180 sketches. Requests were given to subjects in an arbitrary order to intersperse domains and reduce the correlation between sketching styles used in different instances of sketches from the same domain. Sketches were captured using a digitizing LCD tablet.

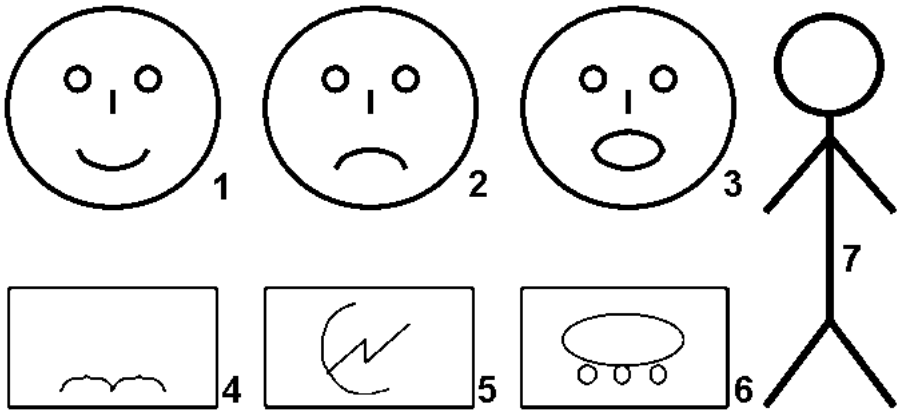


Figure 2-4: Examples of the figures that the users were asked to draw in the user study.

	Object id (from Fig.2-4)						
	1	2	3	4	5	6	7
Number of subparts	5	5	5	6	8	8	6
Max theoretical # of orders	120	120	120	720	40320	40320	720
Mean # of orders used	4	4	3	3	4	3	5

Table 2.1: A summary of the statistics from our user study for a subset of the symbols drawn by the users (shown in Fig.2-4). Note that 120=5!, 720=6! and 40320=8!. Users drew 30 or more examples of each object.

Table 2.1 shows statistics on drawing orders. The maximum possible drawing orders for each object shows the theoretical upper-bound. The table also shows the mean number of drawing orders for all users, rounded to the nearest integer. While in theory there are $n!$ ways of drawing an object with n subcomponents, as Fig. 2.1 shows, only a few are actually employed by users. This confirms our belief that people use predictable stroke orderings.

Our analysis of the sketches also involved constructing sketching style diagrams for each user. A *sketching style diagram* provides a concise way of representing how different instances of the same object are drawn. Nodes of the diagram correspond to partial drawings of an object. They are connected by arcs that correspond to strokes. Fig. 2-5 illustrates the sketching style diagram for the stick-figure example described above.

Our inspection of the style diagrams and the statistics revealed that:

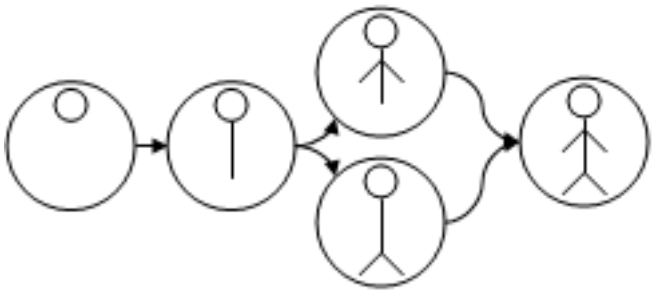


Figure 2-5: A sketching style diagram showing two ways of drawing stick-figures

- People sketch objects in a highly stylized fashion. In drawing the stick figure, for example, one of our subjects always started with the head and the torso, and finished with the arms or the legs (Fig.2-5).
- Individual sketching styles persist across sketches.
- Subjects prefer an order (e.g., left-to-right) when drawing symmetric objects (e.g., the two arms) or arrays of similar objects (e.g., three collinear circles).
- Enclosing shapes are usually drawn first (e.g., the outer circle in emoticons, or the enclosing rectangles in Fig. 2-4).

The user study confirmed our conjecture about the stylized nature of sketching in a number of domains that have received the attention of the sketch recognition community. In order to capitalize on this, we constructed a probabilistic model for learning temporal patterns in sketching and use them for sketch recognition.

Chapter 3

Approach

This chapter describes our approach to sketch recognition. Our model is designed to take advantage of the rich temporal patterns that exist in sketching while supporting interspersed drawing. To facilitate the explanation of this model and to serve as baseline systems, we also introduce two simpler models.

We start by discussing a toy example to illustrate the intuition behind using stroke ordering information for sketch recognition. We then describe properties that we want our models to have and give formal definitions of the terms we will use in the rest of this chapter.

3.1 The intuition

To make the basic intuition clear, we start with an over-simplified task scenario. Assume we have only two symbols to recognize: *skip-audio-track* and *stop*, and assume the user always draws them using the same stroke ordering, indicated by the numbers in Fig. 3-1-a. Finally, assume we need to recognize which of these symbols is present in a scene known to contain only a single instance of one of them (i.e., isolated object recognition).

Suppose the user draws the *stop* symbol as shown in Fig. 3-1-b. Assuming we can reliably recognize the individual strokes as lines and tell whether they are horizontal (**H**), vertical (**V**), negatively/positively sloped (**N**, **P**), we can look at the order in which the user drew the lines and classify the input as a *stop* symbol if we see the

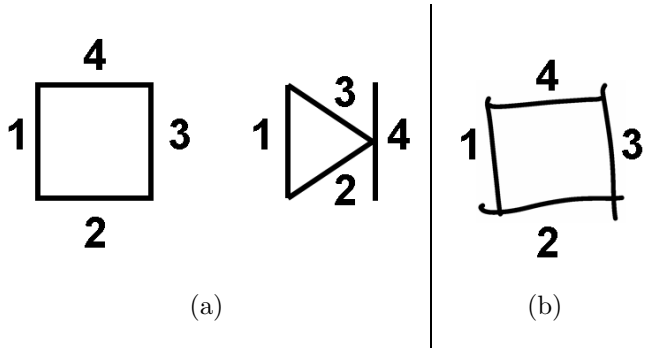


Figure 3-1: Symbols for *stop* and *skip-audio-track* (on the left), and a sketched *stop* symbol (right).

[V, H, V, H] ordering, and as a *skip-track* symbol for the **[V, P, N, V]** ordering.

The above approach works by encoding the user input to generate an *observation sequence* describing the scene (e.g. **[V, H, V, H]**), and comparing this sequence to its model of how the user is known to sketch. The result of the comparison is binary, indicating whether we have a match. This toy example shows how stroke ordering can be used for recognition in an over-simplified scenario.

3.2 Desired features of a model

We see the following properties as being important for models of sketch recognition.

3.2.1 Handling stroke-level patterns

As noted in chapter 2, stroke orderings used in the course of drawing individual objects naturally contain certain patterns. We call these **stroke-level patterns** because they capture the probability of seeing a sequence of *strokes* with certain properties when sketching a particular object. We want our sketch recognition algorithms to capture these patterns.

3.2.2 Support for multiple classes and drawing orders

We should be able to recognize multiple classes of objects. We also want to accommodate multiple drawing orders instead of just one. For example, it may be the case that sometimes the user sketches the *stop* symbol starting with a horizontal line. Further-

more, if the user prefers one drawing order more frequently than others, this should be accounted for as well. This requires using training and classification methods that can use such information.

3.2.3 Handling variations in encoding length

Users should be able to draw freely. For example, they should be able to draw the *stop* symbol using three strokes instead of four, or draw a resistor with five humps instead of six (thus generating an encoding of the input with only five observations instead of six).

3.2.4 Probabilistic matching score

We would like the result of matching an observation sequence against a model to be a continuous value reflecting the likelihood of using that particular drawing order for drawing the object. This is required if we are to have a mathematically sound framework for combining the outputs of multiple matching operations for scenes with multiple objects such that, among plausible interpretations, those corresponding to more frequently used orders are preferred.

3.2.5 Learning compact representations

In practice, different drawing orders will have similar subsequences. Ideally the system should learn compact representations of drawing orders from labeled sketch examples.

3.2.6 Rich feature representation

One of the steps in applying machine learning techniques to a problem is to decide on a set of features that are sufficiently expressive given the problem at hand. In sketch recognition, we deal with data that is most naturally described using geometric features such as the shape of a stroke segment, length and orientation of line segments, radii of circles etc. Some of these features are categorical (e.g., shape of a stroke segment can be `arc`, `line` etc.) and are best represented using discrete variables. Other features such as length and orientation are real-valued quantities and should

be represented as such. Therefore our algorithms should be able to represent both discrete and real valued features.

3.2.7 Handling object level patterns

Another kind of temporal pattern present in online sketches is an **object-level pattern** that captures the probability of seeing a certain sequence of objects being drawn. Consider the domain of UML class diagrams drawn by software designers to describe inheritance relationships, generalizations, associations, etc. In this domain, when a designer draws a new class (indicated by a rectangle), it is natural to expect that the new object will be connected with an arrow to one or more of the objects drawn earlier. So in this domain, it is natural to have arrows drawn after a new class is created, which we define as an *object-level pattern*. It is also natural to expect that the kind of arrow will be different if the newly created class was a `final` class (because `final` classes cannot be extended).

It is easy to imagine that this sort of domain specific knowledge about object-level patterns could be a powerful addition to a UML diagram recognition system, and to sketch recognition systems in general. But not every domain may have patterns that are as well understood as they are for UML diagrams. And even if experts could identify such patterns, incorporating them into a recognition system would be a laborious task at best, considering all the ways in which various objects may combine together. We argue that a better way of incorporating object-level temporal patterns in a recognition framework would be to learn such features from data – along with stroke-level patterns – and use them in recognition. Therefore the ability to model object-level patterns is yet another feature we want.

3.2.8 Ability to handle interspersed drawing

During sketching, although people most often complete each object before starting a new one, as we show later they sometimes draw other objects before completing the current one. Such drawing behavior may severely hinders recognition unless the recognition algorithms deal with it explicitly. The ability to handle interspersed drawing is thus another feature that we desire.

3.3 Terminology and problem formulation

3.3.1 Terminology

We define a ***sketch*** $\mathscr{S} = S_1, S_2, ...S_N$ as a sequence of strokes captured using a digitizer, preserving the drawing order. [1] A ***stroke*** is defined as a set of time-ordered points sampled between pen-down and pen-up events during sketching.

Each stroke is broken into several geometric ***primitives*** such as line and arc segments as part of the preprocessing of the sketch, so let $\mathscr{P} = P_1, P_2, ...P_T$ be the sequence of time-ordered primitives obtained from $\mathscr{S}$. Because the preprocessing of a stroke may result in more than one primitive, the total number of primitives can be larger than the number of strokes.

We use ***segmentation*** to refer to the task of grouping together primitives constituting the same object. Given a set of classes $\mathscr{C} = \{C_1, C_2, ...C_n\}$ ***classification*** refers to the task of determining which object each group of primitives represents (e.g., a stick-figure or a rectangle). Segmentation produces K groups $G = G_1, G_2, ...G_K$, and classification gives us the labels for the groups $L = L_1, L_2, ...L_K$, $L_i \in \mathscr{C}$. Each group is defined by the indices of the primitives included in the group $G_i = \rho_1, \rho_2, ...\rho_n$ sorted in ascending order, so for cases where we don't allow interspersing $|G_i| = \rho_n - \rho_1 + 1$.

We define ***sketch recognition*** as the segmentation and classification of a sketch. A simplifying assumption in most sketch recognition systems is that a stroke can be part of only one object. Our definition of segmentation in terms of primitive groupings is more general than a definition based on stroke groupings and as long as primitives are not shared across objects it allows a stroke to be part of multiple objects (e.g., drawing a box and an arrow, or a resistors and a pair of wires in a single stroke (Fig. 3-2)).

Finally we use the term *observations* to refer to the sequence of features $\mathscr{O} = O_1, O_2, ..., O_T$ obtained from the primitives. We use $\boldsymbol{O_{G_i}}$ to refer to the sequence of observations corresponding to a group of primitives G_i.

[1] We will use the Matlab notation *begin* : *end* as a shorthand for a list of indices that begins with *start* and ends with *end* (i.e., 1:4 = 1 2 3 4). We will also use this notation in subscripts to denote a list of items (i.e., $S_{1:N} = S_1, S_2, ...S_N$).

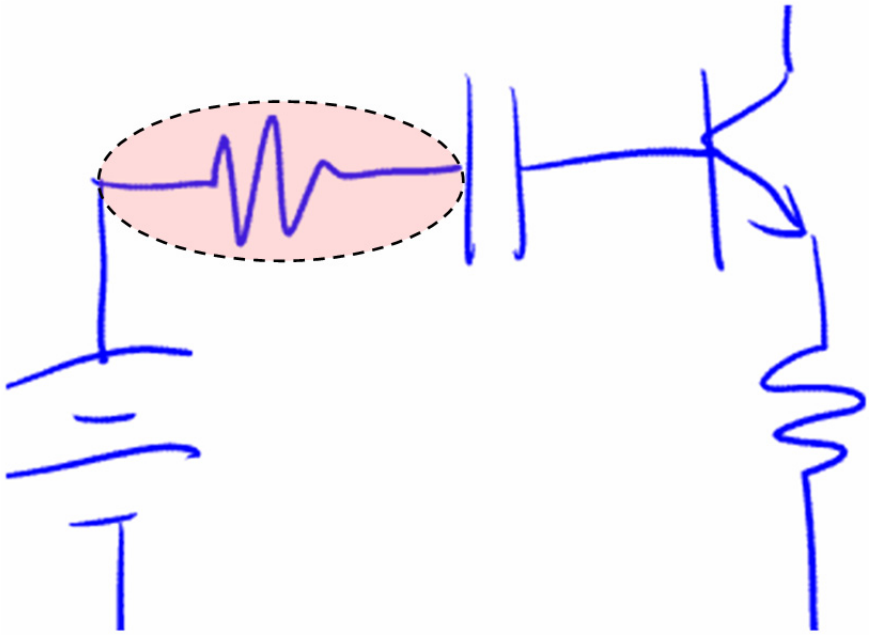


Figure 3-2: An example showing a multi-object stroke. The selected circuit fragment contains two wires and a resistor drawn using a single stroke.

3.3.2 Problem formulation

We consider three models. The first two serve as baseline methods in our evaluations and help introduce our multiscale-interspersing model. Our model can learn both object-level patterns and stroke-level patterns, and it explicitly models interspersed drawing.

Model 1: *Flat model*

This model implements the first five features listed in section 3.2: It can learn compact representations of stroke ordering patterns for multiple objects, each of which can be drawn in more than one way, using a variable number of strokes.

Given a sequence of time-ordered primitives obtained from a sketch, the goal of this model is to find a segmentation and classification of the primitives that maximizes the likelihood of stroke-level patterns. Over all possible ways in which the primitives can be grouped, $\mathfrak{G}$, and all possible ways in which these groups can be labeled, $\mathfrak{L}(G)$, we want the grouping $G \in \mathfrak{G}$ and the labeling $L \in \mathfrak{L}(G)$ that maximizes the likelihood of the stroke-level observable features $\boldsymbol{O}_{\boldsymbol{G_i}}$ for each group G_i given the model corresponding to its label ($\boldsymbol{\lambda}_{\boldsymbol{L(G_i)}}$).

More formally, this can be written as a maximization problem, the solution to which gives us the segmentation and labeling of the input sketch.

$$\underset{G\in\mathfrak{G},L\in\mathfrak{L}(G)}{\operatorname{argmax}} \prod_{i=1}^{K} P(\boldsymbol{O_{G_i}}|\boldsymbol{\lambda_{L(G_i)}}) \tag{3.1}$$

Model 2: *Multiscale model*

This model extends the flat model by adding the ability to have real valued feature vectors and the ability to handle object level patterns in addition to stroke level patterns.

Given a sequence of time-ordered primitives obtained from a sketch, the goal of this model is to find a segmentation and classification of the primitives such that the likelihood of stroke-level patterns and object-level patterns is maximized simultaneously. Over all possible ways in which the primitives can be grouped, $\mathfrak{G}$, and all possible ways in which these groups can be labeled, $\mathfrak{L}(G)$, we want the grouping $G \in \mathfrak{G}$ and the labeling $L \in \mathfrak{L}(G)$ that:

- maximizes the likelihood of the sequence of labels $L_1, L_2, ..., L_K$ given our model for object-level patterns ($\boldsymbol{\lambda_{obj}}$), and
- maximizes the likelihood of the stroke-level observable features $\boldsymbol{O_{G_i}}$ for each group G_i given the stroke-level model corresponding to its label ($\boldsymbol{\lambda_{L(G_i)}}$).

The terms corresponding to the stroke-level and object-level patterns can be written together to obtain the following maximization problem, the solution to which gives us the segmentation and labeling of the input sketch.

$$\underset{G\in\mathfrak{G},L\in\mathfrak{L}(G)}{\operatorname{argmax}} P(L_1, ..., L_K|\boldsymbol{\lambda_{obj}}) \prod_{i=1}^{K} P(\boldsymbol{O_{G_i}}|\boldsymbol{\lambda_{L(G_i)}}) \tag{3.2}$$

The expression above is maximized over the set of all groupings and their labelings ($\mathfrak{G}$ and $\mathfrak{L}$(G)).

Model 3: *Multiscale-interspersing model*

This model adds the ability to handle interspersed drawing to the multiscale model. The formulation of the recognition problem is the same as above, except now the

indices of primitives in each group $\mathfrak{G}$ can have gaps (i.e., $|G_i| > \rho_n - \rho_1 + 1$).

$$\underset{G \in \mathfrak{G}, L \in \mathfrak{L}(G)}{\operatorname{argmax}} \; P(L_1, ..., L_K | \boldsymbol{\lambda_{obj}}) \prod_{i=1}^{K} P(\boldsymbol{O_{G_i}} | \boldsymbol{\lambda_{L(G_i)}}) \tag{3.3}$$

3.4 Choice of probabilistic model and input representation

All of the above formulations require maximizations over the set of all groupings and their labelings ($\mathfrak{G}$ and $\mathfrak{L}$(G)). As noted in Chapter 1, an exhaustive search of this space is computationally prohibitive.

Given that we are modeling sequential patterns, we make the assumption that both stroke-level and object-level patterns can be modeled as products of first order Markov processes. This allows us to efficiently compute maximum likelihood estimates, enabling us to both learn model parameters and do recognition efficiently.

We implement the flat model using a probabilistic model based on *Hidden Markov Models* which is somewhat easier to describe and much simpler to implement compared to the other two models. For the more complex models (#2 and #3), we use *Dynamic Bayesian Networks*. We now briefly review Hidden Markov Models and Dynamic Bayesian Networks mainly for the purposes of introducing notation. Excellent reviews of HMMs and DBNs can be found in Rabiner's HMM tutorial paper [66], and Kevin Murphy's book chapter on DBNs [59]).

3.4.1 Hidden Markov Models

An HMM $\lambda(A, B, \pi)$ is a doubly stochastic process for producing a sequence of observed symbols. An HMM is specified by three parameters A, B, π. A is the transition probability matrix $a_{ij} = P(q_{t+1} = j | q_t = i)$, B is the observation probability distribution $B_j(v) = P(O_t = v | q_t = j)$, and π is the initial state distribution. $Q = \{q_1, q_2, ...q_N\}$ is the set of HMM states and $V = \{v_1, v_2, ...v_M\}$ is the set of observations symbols.

Given an HMM $\lambda(A, B, \pi)$ and a sequence of observations $O = o_1, o_2, .., o_k$, we can efficiently determine how well each model λ accounts for the observations by

computing $P(O|\lambda)$ using the Forward algorithm; compute the best sequence of HMM state transitions for generating O using the Viterbi algorithm; and estimate HMM parameters A, B and π to maximize $P(O|\lambda)$ using the Baum-Welch algorithm.

3.4.2 Dynamic Bayesian Networks

Bayesian networks encode the joint probability of a set of variables $Z = \{Z_1, ..., Z_n\}$ where the graphical structure of the network encodes the conditional dependencies among the variables. DBNs are extensions of Bayesian networks that model joint distribution of a set of variables over time by representing the conditional dependencies between the variables using a pair of Bayesian networks $(B_1, B_{\rightarrow})$. B_1 defines the prior for the Z_i values at time $t = 1$, and $B_{\rightarrow}$ defines how variables at time $t+1$ relate to each other and to those from time t.

3.4.3 The input

The input to our models is the observation sequence $\mathscr{O} = O_{1:T}$ obtained by computing features from corresponding primitives $\mathscr{P} = P_{1:T}$. For the first model, we use only discrete observations, while for the others we support both discrete and real valued (continuous) observations which allows us to have richer features while avoiding discretization problems.

3.5 Flat model

As a reminder, the goal this model is to segment and label an input sketch by finding the solution to the expression in Eq. 3.1, reproduced below:

$$\underset{G \in \mathfrak{G}, L \in \mathfrak{L}(G)}{\operatorname{argmax}} \prod_{i=1}^{K} P(\boldsymbol{O_{G_i}} | \boldsymbol{\lambda_{L(G_i)}}) \tag{3.4}$$

We solve this maximization problem by encoding the sketches using a very simple encoding scheme and modeling stroke orderings using HMMs, then combining the results from individual HMMs using dynamic programming implemented in the form of a shortest path algorithm. By using dynamic programming, we avoid the complexity

of a naïve combinatorial search strategy.

3.5.1 Encoding

We encode strokes using the Early Sketch Processing Toolkit described in [72], which converts strokes into geometric primitives. We use a codebook of 13 symbols to encode the output of the toolkit, converting sketches into discrete observation sequences. Four codebook symbols encode lines: positively/negatively sloped, horizontal/vertical; three encode ovals: circles, horizontal/vertical ovals; four encode polylines with 2, 3, 4, and 5+ edges; one encodes complex approximations (i.e., mixture of curves and lines); and one denotes two consecutive intersecting strokes.

Because instances of the same object sketched in different styles may have encodings of different lengths, we formulated two frameworks for training and recognition that use fixed and variable training examples respectively.

3.5.2 Modeling with fixed input length HMMs

Assume we have $\boldsymbol{n}$ object classes. Encodings of training data for class $\boldsymbol{i}$ may have varying lengths, so let $\boldsymbol{\psi_i} = \boldsymbol{\{l_{i1}, l_{i2}, ...l_{ij}\}}$ be the distinct encoding lengths for class $\boldsymbol{i}$. We partition the training data into $\mathbb{K} = \sum_{i=1}^{n} |\boldsymbol{\psi_i}|$ sets such that each partition has training data for the same object with the same length. Now we train $\mathbb{K}$ HMMs, one for each set, using the Baum-Welch method. Each class $\boldsymbol{i}$ is represented by $|\boldsymbol{\psi_i}|$ HMMs, and we have an inverse mapping that tells us which class each HMM represents.

Assume for a moment that our goal is to do isolated object recognition. Then we could compute $\boldsymbol{P(O|\lambda_i)}$ for each model $\boldsymbol{\lambda_i}$ using the Forward procedure with the observation sequence $\boldsymbol{O}$ generated by encoding the isolated object. $\boldsymbol{\lambda_i}$ with the highest likelihood would give us the object class. But isolated object recognition requires the input sketch to be presegmented, which is usually not the case, and segmentation is a part of the problem we are trying to solve. We achieve both segmentation and recognition by maximizing Eq. 3.4 while noting that for this model, we have assumed the user completes each object before moving to the next one (i.e., there is no interspersed drawing).

Given an observation sequence $\mathscr{O} = \{O_1, O_2, ..., O_T\}$, because we assume there is no interspersed drawing, we can say that prefixes $\mathscr{O}' = \{O_1, O_2, ..., O_{l_{ij}}\}$ of the observation sequence can be accounted for by $\mathbb{K}$ hypotheses – one for each of the $\mathbb{K}$ HMMs that we have. In other words, each HMM model offers a hypothesis that accounts for the first l_{ij} observations with a cost c. Here c is defined as the absolute value of the loglikelihood term obtained for the corresponding prefix $\mathscr{O}'$ and model λ_i written as $c = |log(P(\mathscr{O}'|\lambda_i))|$. Similarly for each of these cases, prefixes of the remaining portions of observation sequence $\mathscr{O}'' = \{O_{l_{ij}}, O_{l_{ij}+1}, ..., O_T\}$ can be accounted for by any of the $\mathbb{K}$ HMMs in a recursive fashion.

We keep assigning hypotheses and scores to subsequences of the observation sequence until we cover all of the observation sequence. Now our task is to find a sequence of hypotheses that account for the whole observation sequence with no gaps or overlaps, such that sum of costs for the hypotheses is minimum. This is essentially equivalent to finding the shortest path in a graph $G(V, E)$ where the nodes correspond to observations and arcs connecting the nodes representing observations O_s, O_d correspond to a hypothesis that accounts for the observations $\{O_s, O_{s+1}, ..., O_d\}$. Once we construct the graph $G(V, E)$, we can find the proper segmentation and labeling of the observations by finding the shortest path and determining which models are actually used in the shortest path.[2]

Here is a step-by-step description of how we build $G(V, E)$. V consists of $|O|$ vertices, one per observation, and a special vertex v_f denoting the end of observations. Let k be the input length for model λ_i. Starting at the beginning of the observation O, for each observation symbol O_s, we take a substring $O_{s,s+k}$ and compute the loglikelihood of this substring given the current model, $log(P(O_{s,s+k}|\lambda_i))$. We then add a directed edge from vertex v_s to vertex v_{s+k} in the graph with an associated cost of $|log(P(O_{s,s+k}|\lambda_i))|$. If the destination index $s + k$ exceeds the index of v_f, instead of trying to link v_s to v_{s+k}, we put a directed edge from v_s to the final node v_f. We set the weight of the edge to $|log(P(O_{s,|O|}|\lambda_i))|$. Here $O_{s,|O|}$ is the suffix of O starting at index s. Adding these special edges from v_s to v_f for $s + k > |O|$ allows our segmentation and recognition algorithms to work even if the user hasn't

[2]The graph $G(V, E)$ that we build for segmentation should not be confused with the graphs that represent HMM topologies.

completed drawing the current object, while preserving global consistency.

This feature is a major strength of our approach, allowing us to to do recognition when the scene is not yet complete – a major challenge in recognition that most other systems sidestep by requiring the user to aid segmentation either implicitly or explicitly. We complete the construction of $\boldsymbol{G}$ by repeating the above operation for all models.

In the constructed graph, having a directed edge from vertex $\boldsymbol{v_i}$ to $\boldsymbol{v_j}$ with cost $\boldsymbol{c}$ means that it is possible to account for the observation sequence $\boldsymbol{O_{i:j}}$ with some model with a loglikelihood of $-\boldsymbol{c}$. The constructed graph may have multiple edges connecting two vertices, each with different costs. By computing the shortest path from $\boldsymbol{v_1}$ to $\boldsymbol{v_f}$ in $\boldsymbol{G}$, we minimize sum of negative loglikelihoods, equivalent to maximizing the likelihood of the observation $\boldsymbol{O}$. The indices of the shortest path gives us the segmentation. Classification is achieved by finding the models that account for each computed segment.

3.5.3 Modeling using HMMs with variable length training data

The formulation above makes the construction of the graph $\boldsymbol{G}$ easy because each HMM is trained using fixed length data. At each step $\boldsymbol{s}$, we can easily compute the destination of the edge originating from the current vertex, $\boldsymbol{v_s}$, by adding the input length for $\boldsymbol{\lambda_i}$ to $\boldsymbol{s}$. One drawback of this method is that it requires an artificial partitioning of the training data for each model, dictated by the variations in description lengths for the same object. This artificial partitioning reduces the total number of training examples per model and prevents representing similar parts of different sketching styles with the same HMM graph fragment, which in turn reduces recognition accuracy and increases cumulative model sizes.

We avoid the artificial partitioning of the training data by grouping the data for all sketching styles together, and training one HMM per object class. After the training is over, for each model we also estimate the probability of ending at each state $\boldsymbol{q}$ of $\boldsymbol{\lambda_i}$ by getting the ending states for the training examples using the corresponding Viterbi paths. This information is used during recognition.

The graph $\boldsymbol{G}$ has the same number of nodes as the previous approach. We generate it by iterating over each model $\boldsymbol{\lambda_i}$, adding edges with the following steps: for each observation symbol $\boldsymbol{O_s}$, we take a substring $\boldsymbol{O_{s,s+k}}$ for each $\boldsymbol{k} \in \boldsymbol{\psi_i}$. Next we compute the loglikelihood for the observation given the current model, $\boldsymbol{log(P(O_{s,s+k}|\lambda_i))}$, and add a directed edge from vertex $\boldsymbol{v_s}$ to vertex $\boldsymbol{v_{s+k}}$ in the graph with an associated cost of $\boldsymbol{|log(P(O_{s,s+k}|\lambda_i))|}$.

We augment each weight in the graph with a term that accounts for the probability that $\boldsymbol{O_{s,s+k}}$ is the encoding of a complete object. This is achieved by penalizing edges corresponding to incomplete objects, by testing whether the observation used for that edge puts $\boldsymbol{\lambda_i}$ in one of its final states using the ending probabilities estimated earlier. Segmentation and recognition is achieved by computing the shortest path in $\boldsymbol{G}$ as described above.

3.5.4 Advantages of the graph based approach

A nice feature of this graph-based approach using dynamic programming is that while the shortest path in $\boldsymbol{G}$ gives us the most likely segmentation of the input, we can also compute the next k-best segmentations using a k-shortest path algorithm[23]. A list of the n-best hypotheses can be used by a user interface that displays them for the user to choose from. This would serve as a means of correcting recognition errors as in speech recognition systems with n-best lists. It is also reasonable to believe that differing portions of the n-best hypotheses correspond to regions of ambiguity, and this information can be used by another algorithm for dealing with ambiguities.

Another nice feature of our approach is that the segmentation algorithm we use is invariant to how the value $\boldsymbol{log(P(O_{s,s+k}|\lambda_i))}$ is computed. Above, each $\boldsymbol{\lambda_i}$ was an HMM that captured temporal patterns but we could as well train two isolated object recognition models $\boldsymbol{\lambda_i^s}$ and $\boldsymbol{\lambda_i^t}$ using spatial and temporal features independently and build a recognition system that uses spatio-temporal information by using $\boldsymbol{log(P(O_{s,s+k}|\lambda_i^{s-t}))}$ instead of $\boldsymbol{log(P(O_{s,s+k}|\lambda_i))}$ where $\boldsymbol{\lambda_i^{s-t}}$ is the joint spatio-temporal model defined by a function $\boldsymbol{\lambda_i^{s-t} = f(\lambda_i^s, \lambda_i^t)}$.

3.6 Multiscale model

This model extends the flat model by adding the ability to have discrete and real valued feature vectors and the ability to handle object-level patterns in addition to stroke-level patterns. Recall that the goal is to segment and label an input sketch by finding the grouping $G \in \mathfrak{G}$ and the labeling $L \in \mathfrak{L}(G)$ that maximizes the expression in Eq. 3.2, reproduced below for reference:

$$\underset{G \in \mathfrak{G}, L \in \mathfrak{L}(G)}{\operatorname{argmax}} \; P(L_1, ..., L_K | \boldsymbol{\lambda_{obj}}) \prod_{i=1}^{K} P(\boldsymbol{O_{G_i}} | \boldsymbol{\lambda_{L(G_i)}}) \tag{3.5}$$

Unlike the previous model, we solve the maximization problem by modeling the observations with a Dynamic Bayesian Network. It models the sketching process as a generative process and tracks its state using a distributed state representation that captures the belief about possible label assignments to each observation and their segmentation over time.

3.6.1 Encoding

One feature of this model is that it allows real-valued observations, and we take advantage of this feature by using real-valued geometric features of the input data. Once again we use the early sketch processing toolkit to preprocess each sketch and obtain a sequence of primitives $P_{1:T}$ (in this case simply line segments). For each primitive P_i, we obtain an observation vector O_t represented as a five-tuple $\boldsymbol{(l_t, \Delta l_t, \theta_t, \Delta\theta_t, sgn_t)}$ where $\boldsymbol{l_t}$ is the length of P_i; $\boldsymbol{\Delta l_t}$ is relative length (l_t/l_{t-1}, 1 for $t = 1$); $\boldsymbol{\theta_t}$ is the angle with respect to the horizontal axis; $\boldsymbol{\Delta\theta_t}$ is the measure of relative angle between P_i and P_{i-1} given by the magnitude of the cross product $\vec{u} \times \vec{v}$ of vectors $\vec{u}$, $\vec{v}$, which in turn are length-normalized versions of P_i and P_{i-1} pointing in the direction of pen movement along each primitive. The only discrete observable $\boldsymbol{sgn_t}$ captures the direction that the stroke turns when moving from P_{i-1} to P_i. It is set to 0 for negative values of $\vec{u} \times \vec{v}$ and 1 for non-negative values (and for the observation at $t = 1$).

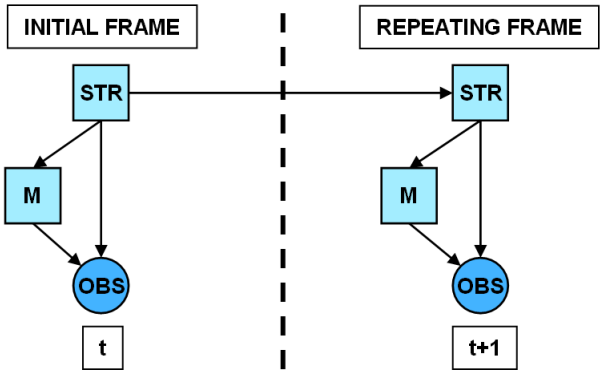


Figure 3-3: The dynamic Bayesian network representing the model for capturing stroke level patterns. This fragment has a dual representation as a hidden Markov model with continuous observations and **M** mixtures.

3.6.2 Model description

Our DBN-based model captures stroke-level and object-level patterns. The model graph has a subgraph that captures the stroke-level patterns. This part of the model conceptually corresponds to the individual object HMMs described in the flat model. For the sake of clarity of presentation, we present our model bottom up, starting with the stroke-level model.

The stroke-level model

For each object class to be recognized, we need a stroke-level model. The purpose of each stroke-level model is to compute the degree of agreement between a particular observation sequence and those sequences typically observed while drawing a particular class of objects.

The stroke-level models answer questions of the sort "what is the likelihood of seeing the observation sequence $\boldsymbol{O}_{\boldsymbol{G}_i}$ obtained from a group of primitives G_i under the rectangle model", or more formally what is $P(\boldsymbol{O}_{\boldsymbol{G}_i}|\boldsymbol{\lambda}_{\boldsymbol{rectangle}})$. This corresponds to the innermost likelihood term $P(\boldsymbol{O}_{\boldsymbol{G}_i}|\boldsymbol{\lambda}_{\boldsymbol{L}(\boldsymbol{G}_i)})$ in expression 3.5. The corresponding model is shown in Fig. 3-3. It has a discrete node **STR** which captures the dynamics of the generative process corresponding to the patterns in the observations provided in the **OBS** variable. The **M** variable is a discrete mixture variable and allows us to represent the observations using mixtures of Gaussians. Each **STR** node is connected to the one in the next time slice, reflecting our choice of modeling stroke-level patterns

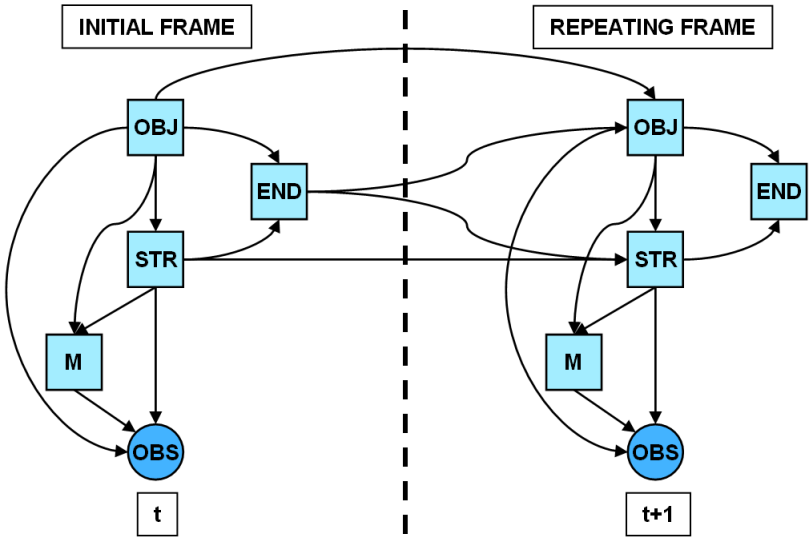


Figure 3-4: The dynamic Bayesian network representing the multiscale model.

as products of a Markov process.

The combined model

To get the final model capturing stroke and object-level patterns, we augment the stroke-level model by two nodes $\mathbf{OBJ_t}$ and $\mathbf{END_t}$ (Fig. 3-4). [3] $\mathbf{OBJ_t}$ is a multi-valued discrete variable that holds our hypothesis of which object P_t is a part of. It can take as many values as the number of classes we can recognize. $\mathbf{END_t}$ reflects our belief about whether the user has just completed drawing object $\mathbf{OBJ_t}$ by drawing the primitive P_t.

The combined model defines a joint distribution over the variables from the stroke-level model, the sequence of object hypotheses $\mathbf{OBJ_{1:T}}$, as well as hypotheses on where each object begins and ends ($\mathbf{END_{1:T}}$). The joint probability over $\mathbf{OBJ_{1:T}}$ corresponds to $P(L_1, ..., L_K|\boldsymbol{\lambda_{obj}})$, the first term term in expression 3.5. Combined with the stroke-level likelihood term $P(\boldsymbol{O_{G_i}}|\boldsymbol{\lambda_{L(G_i)}})$ from above, the combined model allows us to compute the most probable values of $\mathbf{OBJ_{1:T}}$ and $\mathbf{END_{1:T}}$, which gives us the optimal segmentation and labeling of the input sketch.

The value of the **OBJ** node in each time slice is conditioned on the values of

[3] Our combined DBN model has a dual representation as a hierarchical HMM (HHMM) with continuous observables modeled using mixtures of Gaussians. We choose to present the DBN view because DBNs generalize HHMMs and the DBN representation is more efficient.

END and **OBJ** variables from the previous slice. This encodes our belief about the value of $\mathbf{OBJ_{t+1}}$ is based on the values of $\mathbf{OBJ_t}$ and $\mathbf{END_t}$. The justification for this dependence is that, we would expect the value of the **OBJ** node to change only if the user has just finished an object (i.e., $\mathbf{END_t}$=*true*), in which case the next object we would expect to see would depend on what object was just completed. Also note that the three new inter-slice arcs introduced into our model $\mathbf{OBJ_t}\rightarrow\mathbf{OBJ_{t+1}}$, $\mathbf{END_t}\rightarrow\mathbf{STR_{t+1}}$, and $\mathbf{END_t}\rightarrow\mathbf{OBJ_{t+1}}$, cross only a single slice boundary, thus our model remains first order Markovian, enabling training and recognition to be efficient.

In summary, the **OBJ** node keeps track of the class for the objects drawn over time and the joint distribution of the **OBJ** nodes over time gives us $P(L_1, ..., L_K|\ \boldsymbol{\lambda_{obj}})$, the first term in expression 3.5. In the combined model, the distribution of the observations is determined based on the value of the **OBJ** node given by $P(\mathbf{OBS_t}|\mathbf{OBJ_t},\mathbf{STR_t},\mathbf{M_t})$. The choice of which stroke-level process is activated is determined by the **OBJ** node because $\mathbf{STR_{t+1}}$ is conditioned on $\mathbf{STR_t}$ and $\mathbf{OBJ_{t+1}}$.

3.6.3 Implementation issues

There are two issues that need to be addressed in implementing the model described above, both related to the choice of the specific inference algorithm to be used.

As mentioned earlier, our model has a dual representation as a hierarchical hidden Markov model. Unfortunately, the original inference algorithm for HHMMs is both complicated and has time complexity $O(T^3)$ where T is the length of the observation sequence [25]. In our case T is the total number of primitives in a sketch, making the model impractical for situations where real-time feedback is required. By using the DBN representation we can apply efficient graphical model techniques that take linear time [61]. It is therefore essential to use the DBN representation, or convert any HHMM based representation to a DBN prior to inference and learning.

One important implementation issue is numerical instabilities that occur during training, due to the use of continuous observations. Bayesian networks that include continuous and discrete variables (mixed networks) usually represent the continuous variables as Gaussians or mixtures of Gaussians. The conventional belief propagation algorithm used for inference in these networks is the Lauritzen algorithm [52]. Unfortunately this algorithm is susceptible to numerical underflow. Although this has been

documented in the machine learning literature [51, 60], it is not well known because it occurs rarely in practice. The problem appears in the form of singular matrices during inference and is hard to localize. In order to avoid this numerical instability, a specialized algorithm known as the Lauritzen-Jensen stable conditional Gaussian belief propagation should be used [51].

Training

The goal of training is to estimate the parameters of the conditional probability distribution functions for each node in our DBN $(B_1, B_{\rightarrow})$ given a collection of labeled sketches as training data. We use parameter tying to ensure that the probability distribution $P(\mathbf{A}|Parents(\mathbf{A}))$ relating a node $\mathbf{A}$ and its parents is the same for each node A in B_1 and $B_{\rightarrow}$.[4]

During training we know what class each primitive belongs to and we also know when objects begin and end, so the values of the **OBJ** and **END** are observable and are supplied during training. In summary, for each labeled sketch, we use the values $\mathbf{OBJ_{1:T}}$, $\mathbf{END_{1:T}}$ and $\mathbf{OBS_{1:T}}$ to estimate the model parameters.

3.7 Multiscale-interspersing model

The goal of this model is to add the ability to handle interspersed drawing to the multiscale model in a principled way. By interspersing we refer to the situation where the user starts drawing one object but draws one or more other objects before it is completed. For example, Fig. 3-5 shows the case where the user draws the vertical part of the transistor (stroke #2), draws the wire connecting to the collector of the transistor (stroke #3), draws more of the transistor and another wire, and the transistor is actually completed after the current symbol is drawn. In this example two wires (#3 and #6) are interspersed with the transistor.

More formally, suppose we have two objects $\mathscr{A}$ and $\mathscr{B}$. Assume the proper grouping of primitives forming $\mathscr{A}$ and $\mathscr{B}$ are $G_{\mathscr{A}} = \rho_1, \rho_2, ...\rho_m$ and $G_{\mathscr{B}} = \rho'_1, \rho'_2, ...\rho'_n$.

[4] We add dummy parent nodes to the **OBJ** and **STR** nodes in B_1 to ensure that $P(\mathbf{OBJ}|Parents(\mathbf{OBJ}))$ and $P(\mathbf{STR}|Parents(\mathbf{STR}))$ can be represented using conditional probability tables (CPTs) with the same structure for B_1 and $B_{\rightarrow}$.

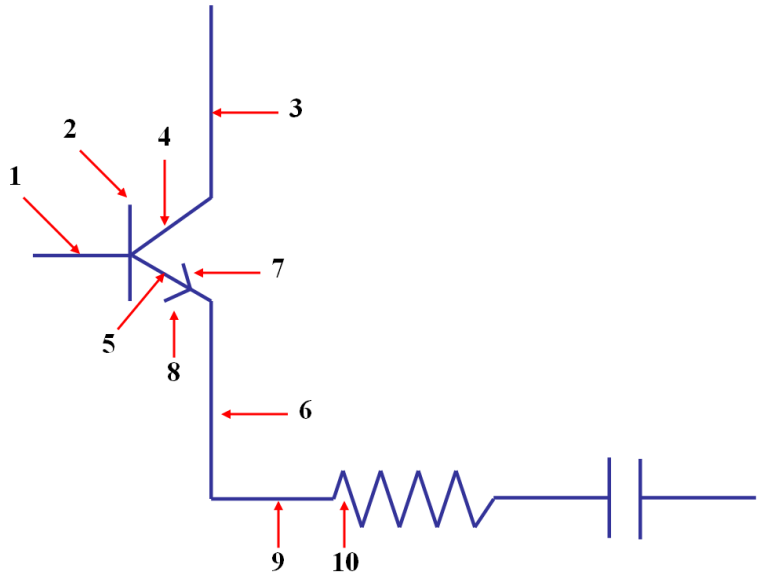


Figure 3-5: An example of interspersing: The user draws two other objects (wires #3 and #6) over the course of drawing the transistor. The drawing order is indicated by numbers.

We say that $\mathscr{A}$ is interspersed with $\mathscr{B}$ if $\rho_1 < \rho'_i < \rho_m$ for $1 \leq i \leq n$ and $|G_{\mathscr{A}}| + |G_{\mathscr{B}}| = p_m - p_1 + 1$. Our model handles a more general case of interspersing where $\mathscr{A}$ can be interspersed with multiple objects.

Interspersing poses a challenge to both models presented earlier. Consider the multiscale model: it defines the state of the sketching process in terms of the current object being drawn $\mathbf{OBJ_t}$, the state of the generative process that captures the stroke level patterns $\mathbf{STR_t}$, and two auxiliary states ($\mathbf{END_t}$ and $\mathbf{M_t}$). The scenario described above where object $\mathscr{A}$ is interspersed with $\mathscr{B}$ requires value of the **OBJ** node to change from $\mathscr{A}$ to $\mathscr{B}$ at time t_1 and back to $\mathscr{A}$ at time t_2 where $\rho_1 < t_1 < t_2 < \rho_m$. At time t_2, the DBN should be able to continue from the state that it would normally assume without interspersing determined by $P(\ \mathbf{STR_t} \mid \mathbf{OBJ_t} = \mathscr{A},\ \mathbf{STR_{t-1}} = s,\ \mathbf{END_{t-1}} = false)$ where $s = \mathbf{STR_{t_1}}$. However state of the **STR** node at time t_1 cannot be recovered because it is overwritten by $\mathbf{STR_{t_1+1}}$ determined by $P(\ \mathbf{STR_t} \mid \mathbf{OBJ_t} = \mathscr{B},\ \mathbf{STR_{t-1}} = s,\ \mathbf{END_{t-1}} = false)$.

It is clear that in order to handle interspersing, state of the process for $\mathscr{A}$ must be saved for resuming later. It is also clear that regardless of the number of classes we support, the user can be drawing only one object at a given time, and only one object process is active at any given time. Based on these two observations, we devised a

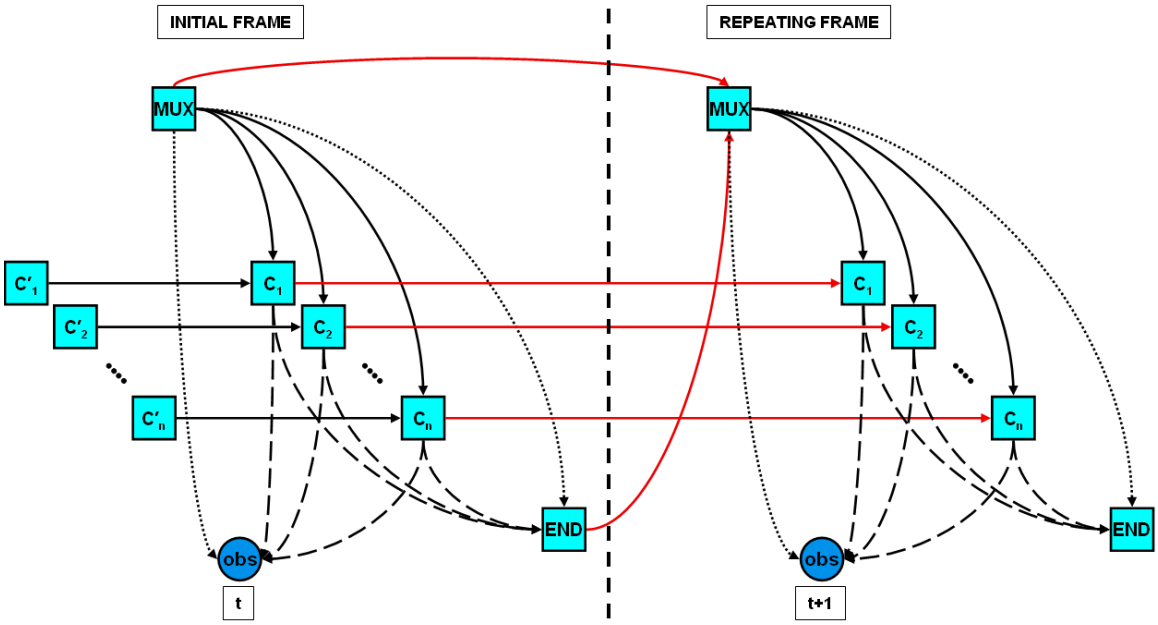


Figure 3-6: The dynamic Bayesian network representing our model that uses multi-scale patterns and handles interspersed drawing.

new model that handles interspersing. As before, we model the sketching process as a generative stochastic process and make use of the *switching parent* mechanism along with parameter tying to model interspersed drawing efficiently while remaining within the formal framework of Dynamic Bayesian Networks. Fig. 3-6 shows our model. We introduce the switching parent mechanism used in our model and then discuss the model in detail.

3.7.1 Switching parents

Our model contains new graphical notation (dotted and dashed arrows in Fig. 3-6) indicating conditional dependencies that change (switch) based on the value of a *switching parent*. The use of *switching parent mechanism* (also known as context specific independence or Bayesian multi-nets) allows us to represent conditional dependencies that change as a function of another node's value in an efficient fashion [15, 28, 37, 16]. Fig. 3-7 shows a simple example of switching parents. In this network **OBS** has two parents **P1** and **P2**, and a *switching parent* **MUX** that controls which one of **P1** or **P2** is activated. The semantics of the network is such that:

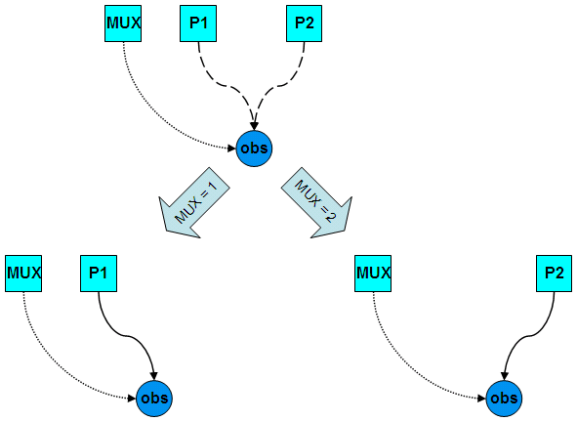


Figure 3-7: Example illustrating the switching parent mechanism.

$$P(\mathbf{OBS}|\mathbf{P1},\mathbf{P2}) = P(\mathbf{OBS}|\mathbf{P1},\mathbf{MUX}=1)P(\mathbf{MUX}=1)+ P(\mathbf{OBS}|\mathbf{P2},\mathbf{MUX}=2)P(\mathbf{MUX}=2)$$

We use switching parents as an efficient mechanism for selecting the process corresponding to the active object in our dynamic model of sketching.

3.7.2 Description of the model topology

The **MUX** node in our model tracks the state of the current object being drawn and the information of what objects are interspersed when interspersing occurs. Cardinality of the **MUX** node is equal to the sum of number the object classes and the number of objects that can be interspersed. The semantics of $\mathbf{MUX_t}= i$ is determined by the following:

$1 \leq i \leq |\mathscr{C}| \Longrightarrow$ drawing object i, $\mathbf{C_i}$ active,

$i > |\mathscr{C}| \Longrightarrow$ interspersing $\mathscr{A}$ and $\mathscr{B}$, given by the function $\mathscr{F}_{int}(i) = \langle \mathscr{A}, \mathscr{B} \rangle$.

The function $\mathscr{F}_{int}(i)$ maps values of **MUX** to a pair of object classes $\langle \mathscr{A}, \mathscr{B} \rangle$. We construct it based on what interspersings exist in the data and which ones we choose to support.

For each object class that we support, the model has a node capturing the dynamics of the stroke-level features (nodes $\mathbf{C}_1$, $\mathbf{C}_2$, ... $\mathbf{C}_{|\mathscr{C}|}$, in Fig. 3-6). As before, features are provided in the **OBS** node and the **END** node is a binary node indicating the user is done drawing the current object. Nodes $\mathbf{C}'_1$, $\mathbf{C}'_2$, ... $\mathbf{C}'_{|\mathscr{C}|}$ are auxiliary nodes for ensuring that the form of the conditional probabilities for

nodes $\mathbf{C}_i$ in the initial frame is the same as those in the repeating frame (e.g., $P_{B_1}(\mathbf{C}_i|Parents(\mathbf{C}_i)) = P_{B_\rightarrow}(\mathbf{C}_i|Parents(\mathbf{C}_i))$. Auxiliary nodes have the same cardinality as their children (i.e., $|\mathbf{C}'_i| = |\mathbf{C}_i|$).

Conditional dependencies for the initial frame

The **MUX** node in the initial frame has no parents and it has a prior that sums to 1 for values corresponding to object classes and 0 for values corresponding to object interspersings. For example a plausible choice for the prior values is to use a uniform distribution:

$$P(\mathbf{MUX_1} = i) = \begin{cases} 1/n & \text{if } 1 \leq i \leq |\mathscr{C}|, \\ 0 & \text{if } i > |\mathscr{C}| \end{cases}$$

The **OBS** node is conditioned on the **MUX** and one of the $\mathbf{C}_i$ nodes as determined by the value of **MUX**. This is an example of the switching parent mechanism. The distribution for **OBS** can be broken into two cases depending on if the user is currently interspersing an object of class $\mathbf{C}_j$ with $\mathbf{C}_k$ given by $\mathscr{F}_{int}(i) = \langle C_j, C_k \rangle$:

$$P(\mathbf{OBS}|\mathbf{MUX} = i, \mathbf{C}_1, \mathbf{C}_2, .., \mathbf{C}_n) = \begin{cases} P(\mathbf{OBS}|\mathbf{MUX} = i, \mathbf{C}_i) & \text{if } 1 \leq i \leq |\mathscr{C}|, \\ & \text{drawing an object of} \\ & \text{type } \mathbf{C}_i \\ P(\mathbf{OBS}|\mathbf{MUX} = i, \mathbf{C}_k) & \text{if } i > |\mathscr{C}|, \\ & \mathscr{F}_{int}(i) = \langle C_j, C_k \rangle, \\ & \text{interspersing } \mathbf{C}_j \text{ with } \mathbf{C}_k \end{cases}$$

The interspersing function $\mathscr{F}_{int}(i)$ mapping values of **MUX** to the range $1 \leq i \leq |\mathscr{C}|$ is provided prior to the training process based on the number of objects we would like to model and the kinds of interspersings present in the training data. Use of switching parents in this fashion also allows us to learn and share a single model for objects that are interspersed (i.e., instead of learning a *wire* model and an *interspersed wire* model, we learn a single *wire* model and reuse it). **END** also has the **MUX** node as its switching parent: $P(\mathbf{END}|\mathbf{MUX} = i, \mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_n) = P(\mathbf{END}|\mathbf{MUX} = i, \mathbf{C}_i)$.

Each $\mathbf{C}_i$ node in the initial frame is conditioned on the **MUX** and $\mathbf{C}'_i$ nodes. The

prior for the $\mathbf{C}'_i$ nodes is represented by a sparse conditional probability table that sets $P(\mathbf{C}'_i = 1) = 1$.[5] This is our way of saying all stroke level processes are at their beginning state when we enter the initial timeslice and based on the value of **MUX**, only one of these nodes updates its state using the inter-frame probability distribution $P_{B_\rightarrow}(\mathbf{C}_{i,t}|Parents(\mathbf{C}_{i,t})) = P_{B_\rightarrow}(\mathbf{C}_{i,t}|\mathbf{C}_{i,t-1},\mathbf{MUX_t})$ substituting the value of $\mathbf{C}'_i$ for $\mathbf{C}_{i,t-1}$. This allows the process selected by **MUX** to change its state from its default *begin* state to a state where it can generate the first observation $\mathbf{OBS_1}$. Using the probability distribution function $P_{B_\rightarrow}(\mathbf{C}_{i,t}|Parents(\mathbf{C}_{i,t}))$ learned over many examples in the first slice of our DBN is another example of parameter tying. $\mathbf{C}_i$ nodes that are not selected by the **MUX** node in the initial frame copy values from $\mathbf{C}'_i$.

Inter-slice dependencies

The **MUX** node at time t is conditioned on $\mathbf{MUX_{t-1}}$ and $\mathbf{END_{t-1}}$. Its value changes based on the object-level transition probabilities if $\mathbf{END_{t-1}}$ indicates that the current observation marks the beginning of a new object (not necessarily of a different class, but a different instance). The **MUX** node can change state even if the $\mathbf{END_{t-1}}$ is $false$ (i.e., $\mathbf{OBS_{t-1}}$ does not mark the end of an object). These cases correspond to interspersings and $P(\mathbf{MUX_t}= i|\mathbf{MUX_{t-1}}= j,\mathbf{END_{t-1}}= false) > 0$ only if have seen objects of type $\mathbf{C}_i$ being interspersed with another object in the training data.[6]

$\mathbf{C_{i,t}}$ nodes in the repeating frames are conditioned on the $\mathbf{MUX_t}$ and $\mathbf{C_{i,t-1}}$ nodes. The CPT for $P_{B_\rightarrow}(\mathbf{C_{i,t}}|Parents(\mathbf{C_{i,t}}))$ is estimated from the data subject to a few constraints that we specify prior to training in the form of deterministic CPTs [14]. Specifically, we require that:

$$P_{B_\rightarrow}(\mathbf{C_{i,t}} = c|\mathbf{MUX_t}= m, \mathbf{C_{i,t-1}} = c') = \begin{cases} 1 & \text{if } m \neq i, 1 \leq m \leq |\mathscr{C}|, c = 1 \\ 0 & \text{if } m \neq i, 1 \leq m \leq |\mathscr{C}|, c \neq 1) \\ 1 & \text{if } m > |\mathscr{C}|, c = c', \text{ and } \mathscr{F}_{int}(m) = \langle C_i, C_* \rangle \\ 0 & \text{if } m > |\mathscr{C}|, c \neq c', \text{ and } \mathscr{F}_{int}(m) = \langle C_i, C_* \rangle \end{cases}$$

[5] Note that we can get away with having only one auxiliary variable $\mathbf{C}'_i$ if all the $\mathbf{C}_i$ nodes have the same cardinality.

[6] For $1 \leq i, j \leq |\mathscr{C}|$ the conditional probability $P(\mathbf{MUX_t}= i|\mathbf{MUX_{t-1}}= j,\mathbf{END_{t-1}}= false)$ is 0 for $i \neq j$, and a non-zero value for $i = j$, thus it can be represented using a sparse conditional probability table.

These constraints ensure that if the user is drawing an object other than the one associated with the node $\mathbf{C_{i,t}}$, its state is reset to the begin state, and if the user has started interspersing an object of type C_i with any other object, the state of the $\mathbf{C_i}$ node is passed on to the next slice. This allows us to save the state of the process associated with C_i so that it can resume after the interspersing is over.

3.8 Discussion

The design of each model we presented was guided by our observations about the domains that we studied including those mentioned in Chap. 2 (finite state machines, UML diagrams, Course of Action Diagrams, stick-figures and emoticons) and circuit diagrams which is our domain of choice for the evaluation chapter (Chap. 4). Among these, the circuit diagrams domain contained many instances of interspersed drawing. In the domains that we have studied, we never observed an object of the same type being interspersed, and our model doesn't handle such cases.

Chapter 4

Results

We report the performance of our multiscale-interspersing model on sketches from the circuit diagram domain to illustrate its performance in absolute terms, and to measure the incremental benefits of modeling object-level patterns and interspersing.

To measure the benefits of modeling object level patterns, we use as a baseline a version of the flat model extended with continuous features and compare its performance to the multiscale model that uses object-level patterns.

To test the benefits of modeling interspersing, we compare the correct recognition rates obtained using the multiscale model to those obtained using the multiscale-interspersing model.

4.1 Circuit data collection

Although anyone can copy a circuit diagram, we believe that professionals who design circuits think about and draw circuit diagrams differently from non-experts. In keeping with this, we collected circuit diagrams from participants studying electrical engineering at MIT, recruiting students who had recently taken the Microelectronic Circuits and Devices course (6.012).

We asked the participants to draw circuits from their course textbook, selecting examples that used a sufficiently expressive and challenging set of circuit components: NPN transistors, resistors, capacitors, batteries and wires. A total of eight participants contributed at least ten circuit diagrams each.

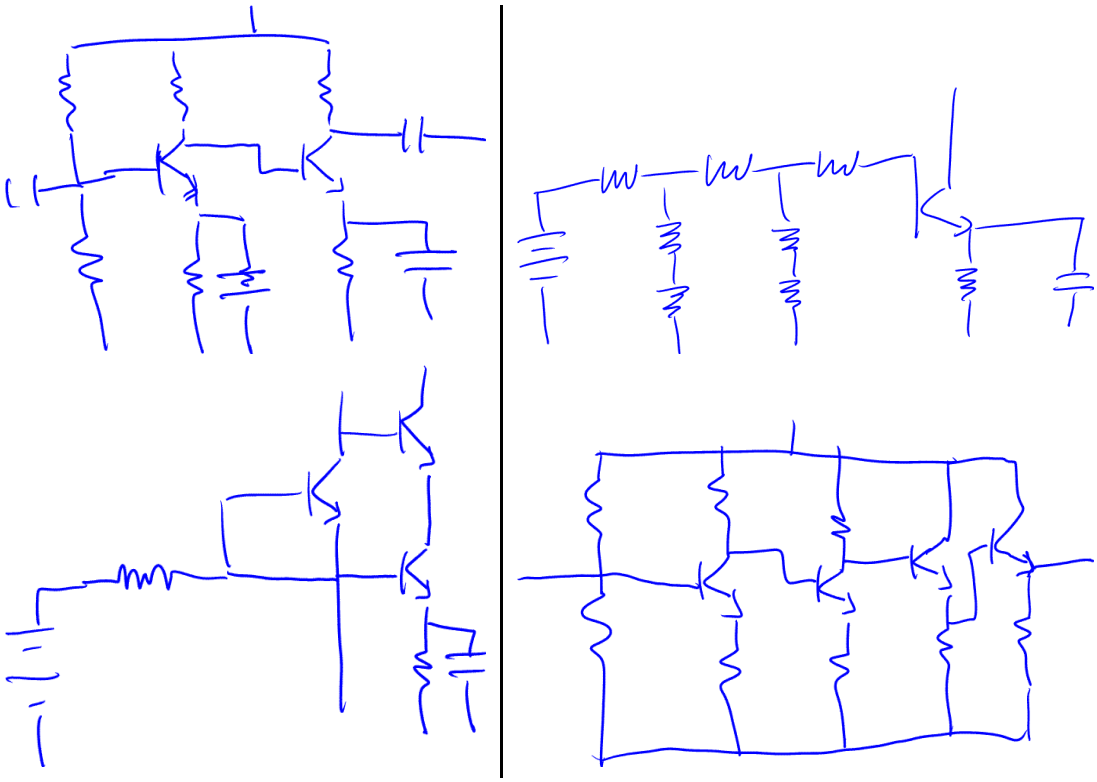


Figure 4-1: Examples of collected circuits.

We began by showing participants diagrams of the circuits to be drawn. To ensure participants got familiar with each circuit, we gave them some time to study each circuit until they understood how it worked. To ensure they understood it, we asked them the function of the circuit (e.g., an inverter circuit, an amplifier), then asked them to explain verbally how it worked. We then removed the textbook diagram and asked them to draw the circuit from memory on a Tablet PC, allowing them to consult the original circuit diagram if needed. Fig. 4-1 shows examples of circuits drawn by the participants. We manually annotated the sketches to be used as training and testing data in our experiments.

4.2 Generating an observation sequence

Both training and classification require converting a sketch to an observation sequence $O_{1:T}$. Using the early sketch processing toolkit, we first preprocess each sketch to obtain a sequence of primitives $P_{1:T}$ (which in this domain contained only line segments). Fig. 4-2 illustrates primitive extraction. The pen direction is indicated with arrows

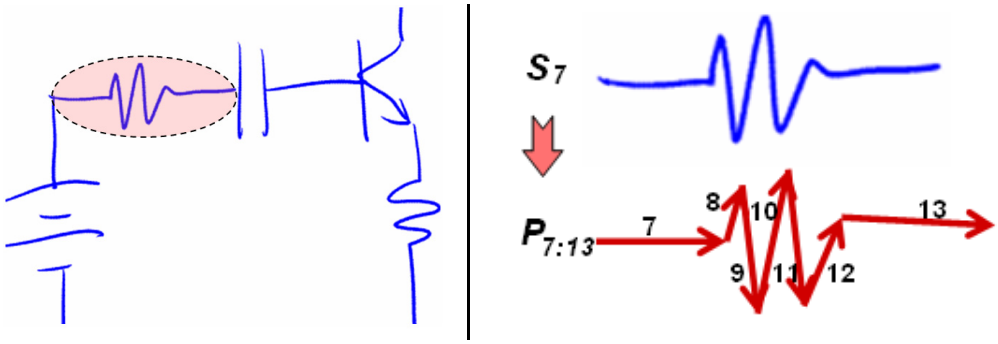


Figure 4-2: Conversion to the primitive sequence.

for each primitive.

For each primitive P_i, we obtain an observation vector O_t represented as a five-tuple $(\boldsymbol{l_t}, \boldsymbol{\Delta l_t}, \boldsymbol{\theta_t}, \boldsymbol{\Delta\theta_t}, \boldsymbol{sgn_t})$ where $\boldsymbol{l_t}$ is the length of P_i; $\boldsymbol{\Delta l_t}$ is relative length (l_t/l_{t-1}, 1 for $t = 1$); $\boldsymbol{\theta_t}$ is the angle with respect to the horizontal axis; $\boldsymbol{\Delta\theta_t}$ is the measure of relative angle between P_i and P_{i-1} given by the magnitude of the cross product $\vec{u} \times \vec{v}$ of vectors $\vec{u}$, $\vec{v}$, which in turn are length-normalized versions of P_i and P_{i-1} pointing in the direction of pen movement along each primitive. The only discrete observable $\boldsymbol{sgn_t}$ captures the direction that the stroke turns when moving from P_{i-1} to P_i. It is set to 0 for negative values of $\vec{u} \times \vec{v}$ and 1 for positive values or $t = 1$.

For our experiments, we create two datasets: "simple" and "mixed". The simple dataset contains only examples without interspersing, while the mixed data contains both the examples from the simple data set and examples where the user interspersed strokes during drawing.

4.3 Modeling object level patterns increases recognition accuracy

4.3.1 Experiment setup and quantitative results

For each user, we ran a series of hold-one-out experiments by getting the recognition rates for each sketch using a model trained on the remaining sketches. For each sketch, we also trained a baseline system which modeled only stroke-level patterns. The baseline system has the same computational complexity as our method and was

	Participant ID							
	1	2	3	4	5	6	7	8
μ_b	83.2	86.5	85.2	73.8	87.1	90.7	79.1	85.0
μ_m	89.4	89.4	87.3	77.4	89.8	93.0	84.6	88.2
Δerr	36.9	21.4	14.1	13.7	20.9	24.7	18.6	21.3
$max\Delta$	65.0	45	31.2	15.0	35.5	40.0	30.7	54.5

Table 4.1: Mean correct recognition rates for the baseline system (μ_b) and the multiscale model (μ_m) on sketches collected from 8 participants. The table also shows the percentage reductions in the error rates and maximum error reductions achieved for each user as percentages (Δ_{err} and $max\Delta$). On average, modeling object-level patterns always improves performance.

obtained by assigning uniform transition probabilities for the object level:

$$P(\mathbf{OBJ_{t+1}}|\mathbf{OBJ_t},\mathbf{END_t}) = \begin{cases} 1/|\mathbf{OBJ}|, & \text{if } \mathbf{END_t} = true \\ 1 & \text{if } \mathbf{END_t} = false \text{ and } \mathbf{OBJ_{t+1}} = \mathbf{OBJ_t} \\ 0 & \text{if } \mathbf{END_t} = false \text{ and } \mathbf{OBJ_{t+1}} \neq \mathbf{OBJ_t} \end{cases}$$

In both models we used three Gaussian mixtures and set the number of states for the stroke-level models to 6. We trained them using simple data (because these models cannot handle interspersing) and tested them on the mixed data (because that is how the users eventually draw).

Table 4.1 shows the average correct recognition rates for each participant obtained using the baseline model and the multiscale model. The table also shows the average and maximum reduction values in the error rates in terms of percentages for each user. As seen in the table, on average, modeling object-level patterns reduced error rates between 14% and 37%, and allowed up to 65% of misrecognition errors to be corrected.

A paired t-test for the values in Table 4.1 showed the difference to be statistically significant for $p \ll 0.05$ and 14 degrees of freedom. We believe such a decrease in the error would substantially improve users' satisfaction with the recognition system in an actual design setting.

4.3.2 Qualitative examples and discussion

We believe it is instructive to explore the kinds of errors avoided by modeling object-level patterns. Consider Fig. 4-3, an amplifier circuit, which also shows the stroke ordering for the fragment of the circuit diagram of particular interest to us. Two interpretations of this circuit are shown in Fig. 4-4.

Fig. 4-4-a shows the interpretation obtained by running the baseline method which encounters four recognition errors. The capacitor **C2** is misrecognized as wires, most probably because of the hook hanging off one of the strokes. The NPN transistor **Q2** is misrecognized because the stroke indicating the current direction is noisy. Note that the vertical part of this transistor (segment #11) and the wire connected to the emitter (segment #12) are grouped together and labeled as a capacitor. Inspection of the sketch reveals that these two strokes have roughly the same length and are sketched one after the other (shown by the numbers in Fig. 4-3). This easily leads to these strokes being mislabeled as a capacitor. [1] Finally, part of the resistor **R1** is misrecognized.

Fig. 4-4-b shows that two of these errors are fixed by the multiscale model. The multiscale model has learned that the chance of a resistor being followed immediately by a transistor, and capacitors being immediately followed by resistors with no wires in between is very small, so these interpretations are penalized. As a result the two misrecognitions are avoided.

4.4 Handling interspersings increases recognition accuracy

4.4.1 Experiment setup and quantitative results

For participants who produced interspersed sketches (#1, #5, #6 and #7) we ran a series of hold-one-out experiments and compared the recognition rate of the multiscale-interspersing model (implemented using features of `GMTK`[14]) and the multiscale model. In both models we used three Gaussian mixtures and set the number of states

[1] This is a result of the limitations of our ordered based recognition.

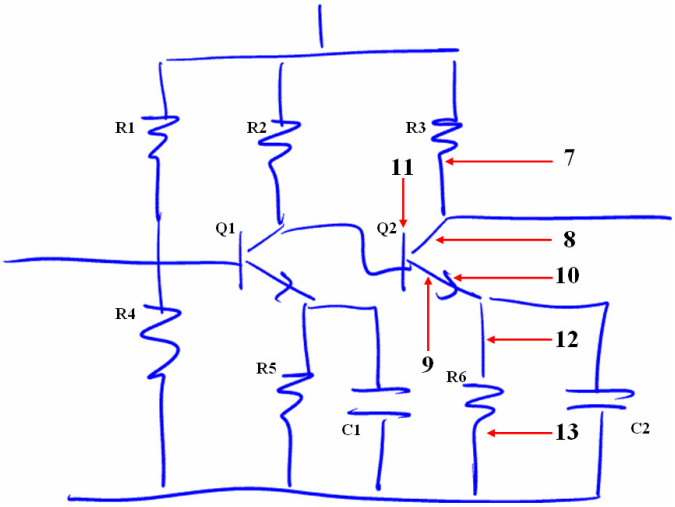


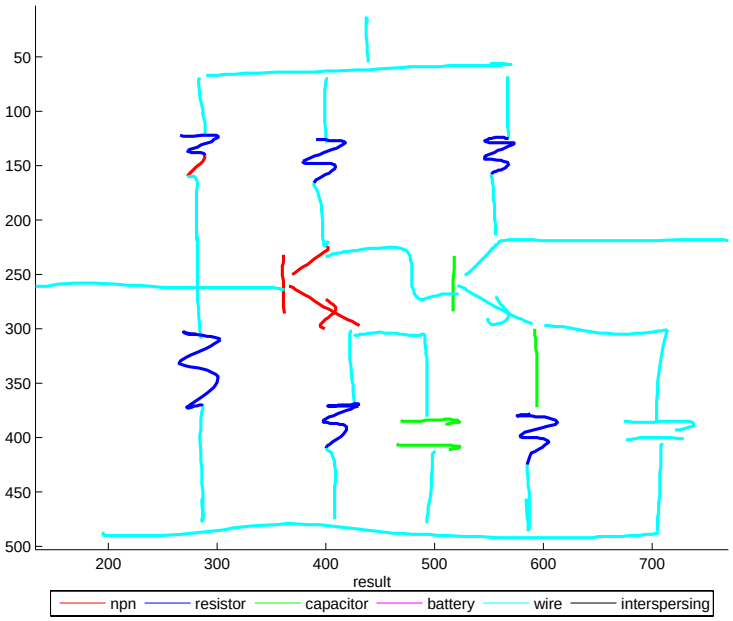
Figure 4-3: One of the circuits drawn by our study participants. Stroke ordering for a fragment of the circuit is indicated by numbers.

	Participant ID			
	1	5	6	7
μ_m	89.4	89.8	93.0	84.6
μ_{m-i}	92.9	92.2	95.6	87.7
Δerr	33.0	23.5	37.1	20.1
$max\Delta$	61.5	33.3	100.0	54.5

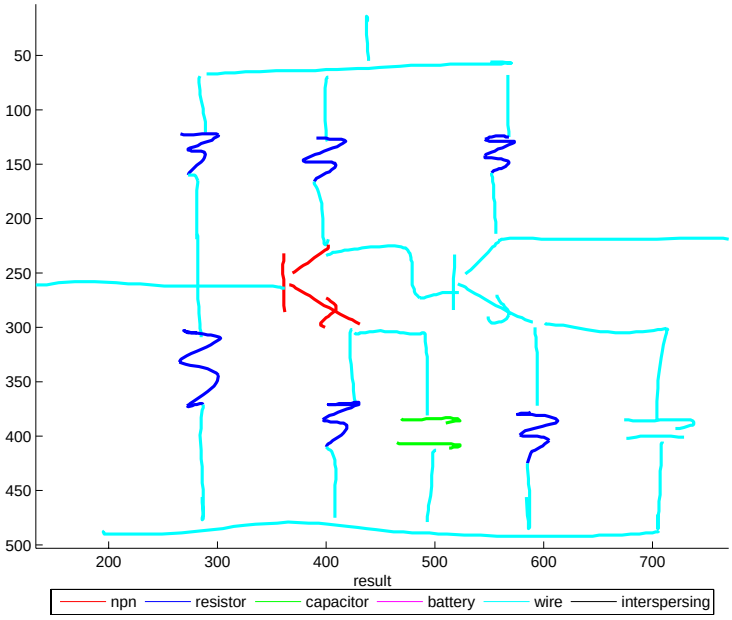
Table 4.2: Mean correct recognition rates for the multiscale-interspersing (μ_{m-i}) and the multiscale models (μ_m) for users who exhibited interspersed drawing behavior. The table also shows the percentage reductions in the error rates and maximum error reductions achieved for each user as percentages (Δ_{err} and $max\Delta$). On average, handling interspersing patterns always improves performance.

for the stroke-level models to 6. The DBN representing the multiscale-interspersing model is shown in Fig. 4-5. We trained the multiscale model using simple data (because it cannot handle interspersing) and the multiscale-interspersing model using the mixed data. We tested both models using mixed data.

Table 4.2 shows the average correct recognition rates for each participant obtained using the multiscale model and the multiscale-interspersing model. The table also shows the average and maximum reduction values in the error rates in terms of percentages for each user. As seen here, on average, handling interspersing improves performance, and allows 20%-37% of misrecognition errors to be corrected.



(a) Interpretation of the baseline model.



(b) Interpretation of the multiscale model.

Figure 4-4: Examples of errors corrected using context provided by object-level patterns. There are four misrecognitions in (a). Note how two of these misrecognitions is fixed using knowledge of object level patterns (resistor in the upper left corner (**R1**) and the wire connected to the emitter of the misrecognized transistor).

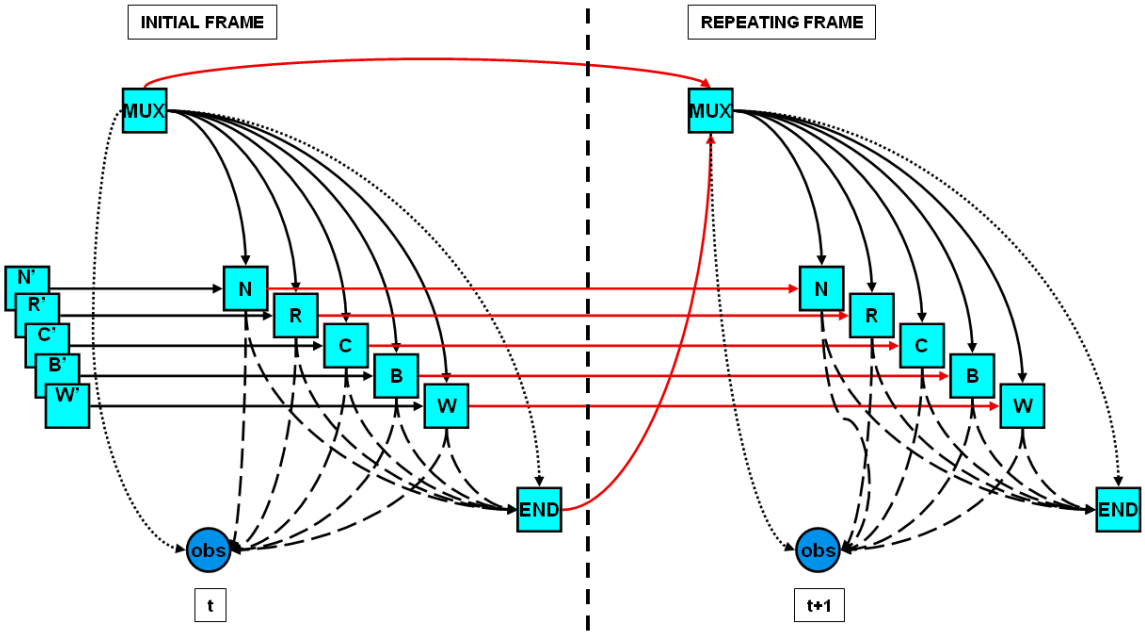


Figure 4-5: The dynamic Bayesian network representing the multiscale-interspersing model that handles interspersed drawing.

A paired t-test for the values in Table 4.2 found the difference to be statistically significant for $p < 0.05$ and 6 degrees of freedom.

4.4.2 Quantifying errors per-interspersing

The percentages presented above do not show the full extent of the negative effects of not handling interspersing because the percentage of errors due to interspersing is diluted by the large number of other components successfully recognized. A more informative measure of the benefits of handling interspersing can be obtained if we look at the number of misclassified primitives per interspersed primitive.

In order to measure this, we ran both the multiscale model (the baseline) and the interspersing-multiscale model on a control group that contained ten sketches with no interspersing, and a test group that contained versions of these sketches with interspersings. The interspersed version of each sketch was obtained by moving one of the wires preceding/following a transistor back/forward in time. Our use of these examples ensures that we measure the misrecognition effects due only to interspersing.

We trained our baseline model with the non-interspersed data and supplied the interspersing-multiscale model with the additional interspersed examples. Table 4.3

	Sketch ID										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	μ
Total primitives	90	85	104	101	92	100	87	105	80	98	94.2
Added interspersings	2	2	4	4	3	3	2	2	2	4	2.8
Missed by the baseline	7	9	15	12	12	12	13	13	11	19	12.3
Missed by our model	3	3	4	1	0	0	2	2	2	3	2
Difference	4	6	11	11	12	12	11	11	9	16	10.3
Missed prim./intersp.	2	3	2.75	2.75	4	4	5.5	5.5	4.5	4	3.8

Table 4.3: This table shows the number of misrecognized primitives per interspersing. The first two row shows the number of total primitives in each sketch and the number of added interspersings. Next two rows show the number of primitives incorrectly recognized by the baseline and by our model. The next row shows the number of primitives that the baseline misses because it cannot handle interspersings and the last row is the number of primitives missed by the baseline per interspersing.

shows the number of misrecognized primitives per interspersing. The first two rows show the number of total primitives and the number of added interspersings per sketch. The table also shows the total number of primitives incorrectly recognized by the baseline and by our model. The last two rows show the total of primitives that the baseline misses because it cannot handle interspersings and the number of primitives missed by the baseline per interspersing. As seen in the table, a single interspersing causes as many as 16 primitives to be misclassified. Considering a transistor has five primitives, that is worth about three transistors. When we normalize the number of misrecognized primitives per sketch by the number of interspersings we get about 2-6 misrecognized primitives per interspersing.

In the best case, the errors caused by each interspersing will require at least one correction by the user (e.g., when all the misrecognized shapes belong to the same shape). In the worst case the errors may require as many corrections as the number of misclassified primitives, further showing the utility of modeling interspersing.

4.4.3 Qualitative examples and discussion

Fig. 4-6 shows an example illustrating how interspersed drawing causes misrecognitions if not handled properly. This example is particularly instructive because it shows that interspersing only a single primitive can lead to a cascade of misrecog-

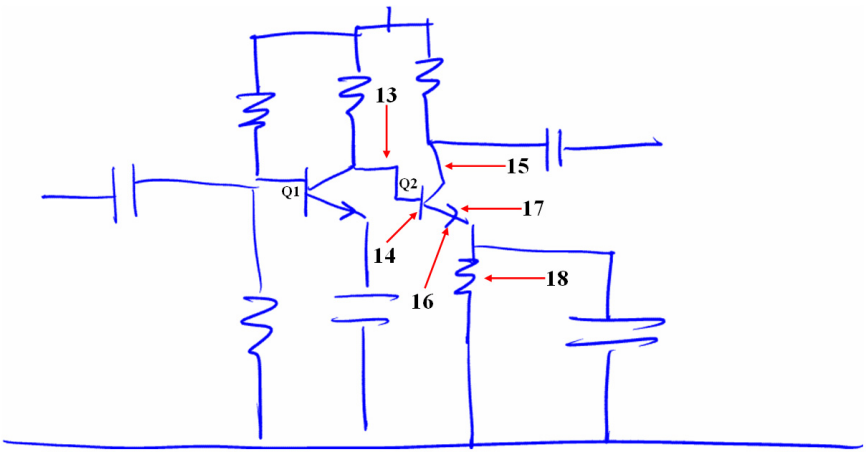


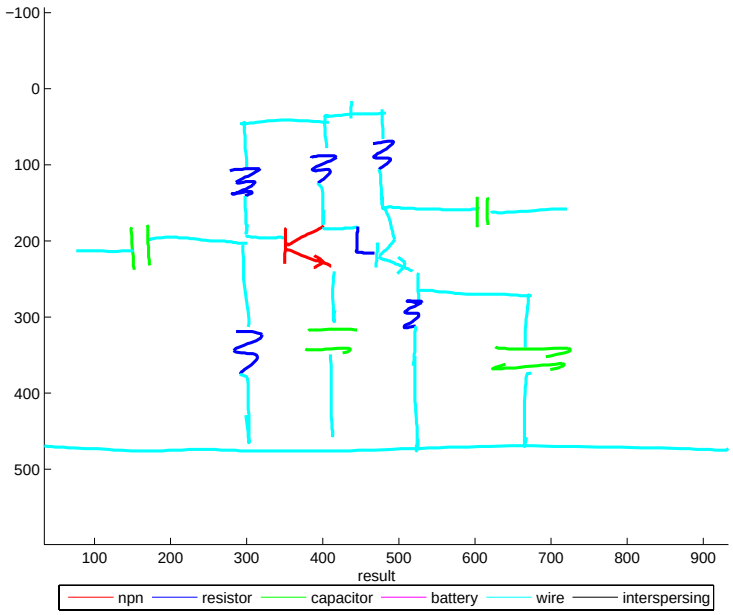
Figure 4-6: Another circuit drawn by our study participants. Stroke ordering for a fragment of the circuit is indicated by numbers.

nitions. In this example, the user drew the collector of the transistor **Q2** and the wire connected to it using a single stroke (stroke #15, which is also an example of multi-object stroke).

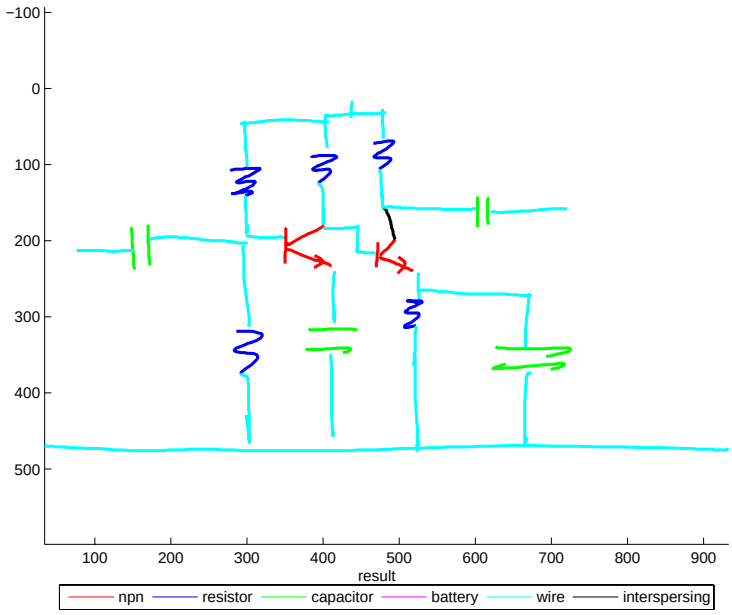
Two interpretations of the circuit are shown in Fig. 4-7. Fig. 4-7-a shows the interpretation obtained by running the multiscale model. In this figure, there are two recognition errors. **Q2** is misclassified as a set of wires, and the two wire segments connected to the base are misclassified as a resistor.

Fig. 4-7-b shows that both errors are fixed when we use the multiscale-interspersing model. The multiscale-interspersing model has learned that with probability 0.14 wires can be drawn in the course of drawing a transistor. This not only allows the transistor to be identified correctly but also helps the two wire segments to be classified correctly by using the knowledge that transistors very rarely proceed resistors, $P(\mathbf{OBJ_t}{=}\mathbf{NPN} \mid \mathbf{OBJ_{t-1}}{=}\mathbf{RESISTOR}) \approx 0$. This example shows the benefits of modeling both object-level patterns and interspersing.

The incremental benefits of using a more elaborate model may appear at times to be small (e.g. Fig 4-8). Nevertheless, correcting each misclassification requires effort on the part of the user and gets in the way of getting the job done. Hence we believe the error reduction rates we have demonstrated are significant in the context of a sketch-based user interface.

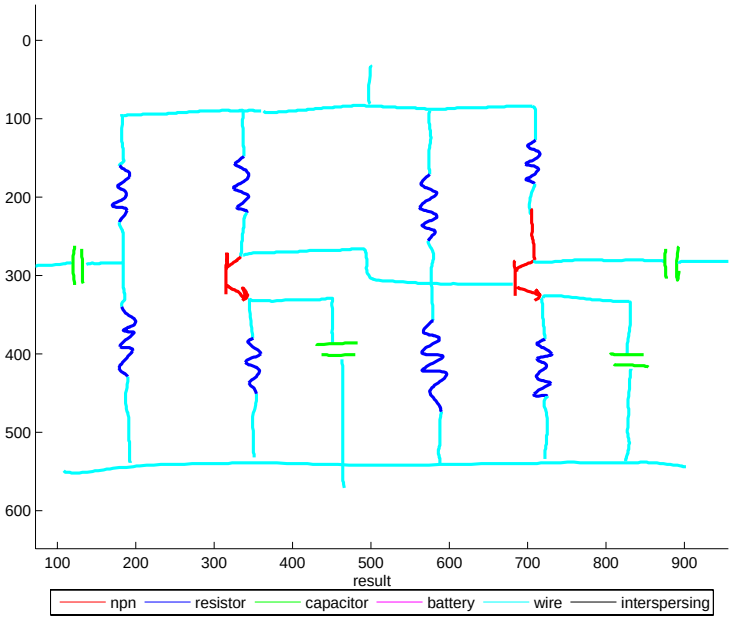


(a) Interpretation of the multiscale model.

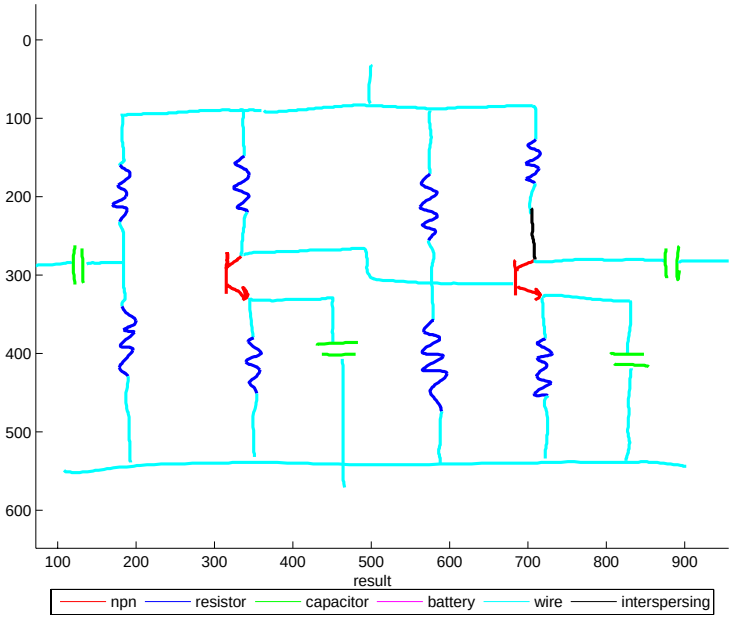


(b) Interpretation of the multiscale-interspersing model.

Figure 4-7: In this example, the user interspersed a wire connected to the collector of the transistor on the right. As a result the transistor is misclassified by the multiscale model (a). The multiscale-interspersing model identifies the wire interspersing (indicated by black) and correctly interprets the transistor (b).



(a) Interpretation of the multiscale model.



(b) Interpretation of the multiscale-interspersing model.

Figure 4-8: Another example of handling interspersing. In this case the errors caused by the interspersing is minor compared to 4-7. The interspersed wire is interpreted to be part of the transistor. The multiscale-interspersing model identifies the wire interspersing (indicated by black in b).

4.5 Summary

We have demonstrated that it is possible to recognize sketches using temporal patterns that naturally appear during sketching. This is particularly noteworthy because until recently time-based graphical models were considered to be unsuitable for sketch recognition [5].

We also showed that using object-level temporal patterns increases recognition rates by incorporating contextual information about object orderings. We reported that using a naïve implementation of belief propagation in the context of our graphical model is susceptible to numerical instabilities and suggested that the Lauritzen-Jensen stable conditional Gaussian belief propagation should be used to avoid them. Finally we showed that we can explicitly model interspersed drawing behavior in a principled way. This allowed us avoid cascades of misinterpretations often caused by interspersings.

Finally, we note that the baseline models that we described are easier to implement compared to our main model. This gives designers the choice to work with easier to implement methods at the cost of sacrificing expressiveness of the model and recognition accuracy.

Chapter 5

Related work

5.1 Sketch based interfaces

[38, 18, 48, 42, 41, 4, 5, 27, 34, 2, 11, 33, 43, 82, 12, 64, 35, 40, 49, 26, 22, 1, 3, 63, 30, 31, 84, 62, 69, 21, 10, 24, 86, 13, 32, 50, 36]

5.2 Recognition systems

5.2.1 Graphics recognition

[85, 9, 77]

5.2.2 Handwriting recognition

[47, 65, 57]

5.2.3 Symbol recogntion systems

[68, 39, 58, 45, 83, 7, 19]

5.3 Sketch recognition systems

Work in sketch recognition can be divided into two groups based on whether it uses temporal features. We start with a review of the large body of work that does not use

temporal features. These systems are complementary to our approach. Then discuss those that use temporal features for recognition.

5.3.1 Use of spatial data for recognition

[79][4, 5, 29, 44, 46, 75, 73, 74, 70, 71, 55, 56, 6, 53, 54, 17, 20]

5.3.2 Use of temporal data for recognition

The handwriting recognition community has come up with a large number of methods to recognize single stroke or segmented pen input using HMMs. In the sketch recognition community, work in [8] describes a similar HMM-based symbol recognizer that uses chain-code-like features to recognize isolated symbols. Our work does not assume segmented input and builds a joint model for complete sketches.

In [76], the authors present a sketch interpretation and curve refinement system that uses an HHMM setup. This work shares the same motivation as ours. In their work, they assume that the parameters of the object-level process (which they refer to as the scene level) is supplied by the user using a *semantic graph* representation. We learn the parameters of the stroke-level and object-level patterns from data, and use the more efficient DBN representation to avoid the $O(T^3)$ complexity of HHMMs. Furthermore, we support multi-stroke objects and real-valued continuous observations which allow using a rich set of features.

Chapter 6

Contributions

6.1 Contributions

We have presented a sketch recognition framework that exploits both stroke-level and object-level temporal orderings. Our framework allows learning object-level patterns from data, handles multi-stroke objects and multi-object strokes efficiently, supports continuous observable features and handles interspersed drawing.

A number of contributions make our work unique:

- Our work is the first to demonstrate the utility of stroke ordering information and the suitability of time-based graphical models for recognizing complicated sketches and is particularly noteworthy because until recently such models were considered to be unsuitable for sketch recognition [5].

- We showed how graphical models can be used to implement our model of temporal patterns efficiently using features such as switching parents, sparse representations and parameter tying. Our model interprets sketches with several dozen objects under a second on a standard PC, and as such is suitable for real-time sketch interpretation.

- We reported that a standard implementation of belief propagation, appropriate for simple conditional Gaussian belief networks, runs into numerical instabilities when used for inference in our model. We noted that the Lauritzen-Jensen stable

conditional Gaussian belief propagation algorithm should be used to avoid such numerical problems in recognition.

- We demonstrated that our hierarchical models capture knowledge of both common stroke orderings and common object orderings. Our evaluation involving circuit diagrams collected from eight participants shows that modeling common object orderings reduces the number of recognition errors by 14%- 37%.

- We also showed how a probabilistic temporal model can explicitly model interspersed drawing behavior. Interspersed drawing behavior poses serious challenges to a naïve approach of temporal sketch recognition model, but we address the issue remaining within the formal framework of Dynamic Bayesian Networks. Therefore we do not rely on the the assumption that users to draw objects one at a time. We showed that modeling interspersing reduces recognition errors by an additional 20%-37%.

- Our recognition framework allows strokes to be part of multiple objects as long as objects do not share primitives. We support multi-stroke objects and objects with variable number of components (e.g., resistors with variable number of humps).

- Finally we support discrete and real-valued feature representations that allows us to encode our data using discrete encodings for categorical properties (e.g., a stroke being a circle vs. a line), and real-valued encodings for continuous features (e.g., length, orientation of a line).

6.2 Discussion

Our recognition framework also has a number of limitations that make interesting future research topics. Although our user studies and psychology research suggests that drawing is a highly stylized process and ordering phenomena is found in many domains [78, 81], our approach is not suitable for domains where stroke ordering is unpredictable.

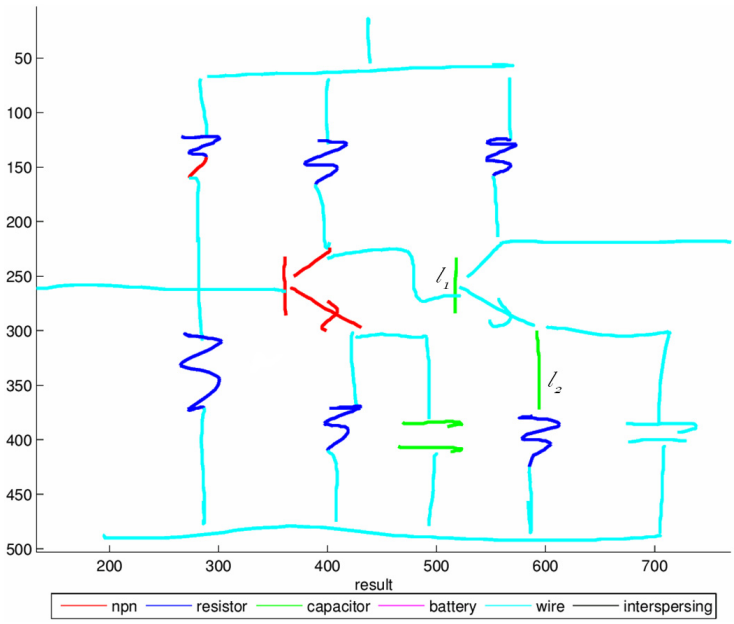


Figure 6-1: An example of strokes with the right temporal character but wrong shape being classified incorrectly (reproduced from Chap. 4). The two vertical lines (l_1, l_2) are offset from one another, but were classified to be a capacitor because they were drawn one after the other, and the temporal features extracted from them agree with the capacitor model that we learned.

Another limitation of our model is that currently it does not incorporate any spatial or geometric constraints beyond those used to encode stroke sequences, and as a result recognition is based strictly on temporal patterns and not shape. Therefore a sequence of strokes that has the right temporal character but the wrong shape can be classified incorrectly. For example, the two parallel strokes in Fig. 6-1 were classified as a capacitor, even though the strokes were offset from one another. We consider below how this limitation can be addressed by using shape based classifiers as external sources of information.

Chapter 7

Future work

7.1 Incorporating spatial features

Hypotheses supplied by external recognizers that use spatial or structural information can be incorporated into our model using virtual evidence nodes [67]. This would be a first step toward combining temporal and spatial/structural information for sketch recognition.

7.2 Exploiting multi-scale feature representations

7.3 Handling handwriting

Bibliography

[1] James A. Landay A. Chris Long, Jr. and Lawrence A. Rowe. 'those look similar!' issues in automating gesture design advice. *PUI*, November 15-16, 2001.

[2] Aaron Adler. Segmentation and alignment of speech and sketching in a design environment. *Masters Thesis, Cambridge, MIT*, February 2003.

[3] Christine Alvarado. A natural sketching environment: Bringing the computer into early stages of mechanical design. Master's thesis, Massachusetts Institute of Technology, 2000.

[4] Christine Alvarado. Sketchread: A multi-domain sketch recognition engine. In *Proceedings of ACM Symposium on User Interface Software and Technology 2004*, 2004.

[5] Christine Alvarado and Randall Davis. Dynamically constructed bayes nets for multi-domain sketch recognition. In *Proceedings of IJCAI 2005, California, August 1 2005*.

[6] Christine Alvarado, Mike Oltmans, and Randall Davis. A framework for multi-domain sketch recognition. *AAAI Spring Symposium: Sketch Understanding*, 2002.

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

[8] D Anderson, C Bailey, and M Skubic. Hidden markov model symbol recognition for sketch-based interfaces. *AAAI Fall Symposium Series, Making Pen-Based Interaction Intelligent and Natural*, 2004.

[9] H. S. Baird, H. Bunke, and K. Yamamoto. Structured document image analysis. Springer-Verlag, Heidelberg, 1992.

[10] M. Banks and E. Cohen. Realtime spline curves from interactively sketched data. In *SIGGRAPH, Symposium on 3D Graphics*, pages 99–107, 1990.

[11] Song Baohua, Lu Changde, and Yang Haicheng. Research on product-tree based multimode and integrated product model. *In Proceeding of the 6th International Conference On Manufacturing Technology, Hong Kong.*, Dec 16-18, 2001.

[12] Song Baohua, Tang Mingxi, JH Frazer, and Yang Haicheng. Stroke-based intelligent sketching interface. *Proceeding of the 5 th Asia Pacific Conference on Computer Human Interaction (APCHI2002)*, pages 500–509, Nov. 2002.

[13] A. Bentsson and J. Eklundh. Shape representation by multiscale contour approximation. *IEEE PAMI 13, p. 85–93, 1992.*, 1992.

[14] J. Bilmes and G. Zweig. The graphical models toolkit: An open source software system for speech and time-series processing. *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing. Orlando Florida*, 2002.

[15] Jeff .A. Bilmes. Dynamic bayesian multinets. *In Proceedings of the 16th conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann*, 2000.

[16] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. *Uncertainty in Artificial Intelligence*, 1997.

[17] Mike Oltmans Christine Alvarado and Randall Davis. A framework for multi-domain sketch recognition. *AAAI Spring Symposium: Sketch Understanding*, 2002.

[18] Philip Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow. Quickset: Multimodal interaction for distributed applications. *ACM MM 1997*, 1997.

[19] Manuel J. da Fonseca Csar F. Pimentel and Joaquim A. Jorge. Experimental evaluation of a trainable scribble recognizer for calligraphic interfaces. *Proceedings of the Fourth Int. Workshop on Graphics Recognition, (GREC'01), Ontario, Canada*, 2001.

[20] Randall Davis. Sketch understanding in design: Overview of work at the mit ai lab. *AAAI Spring Symposium: Sketch Understanding*, 2002.

[21] L. Eggli. Sketching with constraints. Master's thesis, University of Utah, 1994.

[22] Jacob Eisenstein. *placeholder entry.*

[23] David Eppstein. Finding the k shortest paths. *SIAM J. Computing Vol.28, No.2, pp. 652-673*, February 1988.

[24] T. Igarashi et. al. Interactive beautification: A technique for rapid geometric design. In *ACM Symposium on User Interface Software and Technology*, pages 105–114, 1997.

[25] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning 32:41*, 1998.

[26] Kenneth D. Forbus, Ronald W. Ferguson, and Jeffery M. Usher. Towards a computational model of sketching. In *Intelligent User Interfaces*, pages 77–83, 2001.

[27] Andrew S. Forsberg, Mark Dieterich, and Robert C. Zeleznik. The music notepad. In *ACM Symposium on User Interface Software and Technology*, pages 203–210, 1998.

[28] D. Geiger and D. Heckerman. Knowledge representation and inference in similarity networks and bayesian multinets. *Artificial Intelligence, 82:4574*, 1996.

[29] Leslie Gennari, Levent Burak Kara, and Thomas F. Stahovich. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *AAAI Fall Symposium Series, Making Pen-Based Interaction Intelligent and Natural*, 2004.

[30] M. Gross. Recognizing and interpreting diagrams in design. In *2nd Annual International Conference on Image Processing*, pages 308–311, 1995.

[31] M. Gross and E. Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of ACM Symposium on User Interface Software and Technology 96*, pages 183–192, 1996.

[32] Tracy Hammond. A domain description language for sketch recognition. *Proceedings of 2002 SOW*, 2002.

[33] Tracy Hammond and Randall Davis. Automatically transforming shape descriptions for use in sketch recognition. *AAAI 2004*, 2004.

[34] Tracy Hammond and Randall Davis. Ladder: A language to describe drawing, display, and editing in sketch recognitio. *Proceedings of IJCAI*, August 2003.

[35] Song Han and Grard Medioni. 3dsketch: modeling by digitizing with a smart 3d pen. *International Multimedia Conference, Proceedings of the fifth ACM international conference on Multimedia*, pages 41 – 49, 1997.

[36] M. Hearst. Sketching intelligent systems. *IEEE Intelligent Systems*, pages 10–18, May/June 1998.

[37] D. Heckerman. Probabilistic similarity networks. *MIT Press*, 1991.

[38] Christopher Herot. Graphical input through machine recognition of sketches. *Proceedings of the 3rd annual conference on Computer graphics and interactive techniques, pp. 97 - 102*, 1976.

[39] Heloise Hse and A. Richard Newton. Recognition and beautification of multistroke symbols. *AAAI Fall Symposium Series, Making Pen-Based Interaction Intelligent and Natural*, 2004.

[40] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. *SIGGRAPH 99*, pages 409–416, August 1999.

[41] Hong I. J., Landay A. J., Long A. C., and Mankoff J. Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. *AAAI Sketch Symp. March 25-27*, 2002.

[42] Forbus K., Ferguson, and Usher J. Towards a computational model of sketching. *Proceedings of IUI'01. Santa Fe, NM.*, 2001.

[43] Ed Kaiser, David Demirdjian, Alexander Gruenstein, Xiaoguang Li, John Niekrasz, Matt Wesson, and Sanjeev Kumar. Demo: A multimodal learning interface for sketch, speak and point creation of a schedule chart. *Proceedings of the Sixth International Conference on Multimodal Interfaces (ICMI 2004), State College, Pennsylvania, USA, October 14-15, 2004, pgs. 329-330.*

[44] Levent Burak Kara and Thomas F. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. *AAAI Fall Symposium Series, Making Pen-Based Interaction Intelligent and Natural*, 2004.

[45] Kunihiro Kondou, Naoki Kato, and Toshinori Watanabe. Osr:online sketch recognition by data compression technique. *IPSJ JOURNAL Abstract Vol.38 No.12 - 007 pp. 2468-78*, 1997.

[46] Balaji Krishnapuram, Christopher Bishop, and Martin Szummer. Generative bayesian models for shape recognition. *IWFHR '04, Japan*, 2004.

[47] Schomaker Lambert. From handwriting analysis to pen-computer applications. *IEEE Communication Eng., 10(3), pp. 93-102*, 1998.

[48] A. James Landay and Brad Myers. Sketching interfaces: Toward more human interface design. *IEEE Comp., 34-43 2001*, 2001.

[49] J.A. Landay, J.I. Hong, S.R. Klemmer, J. Lin, and M.W. Newman. Informal puis: No recognition required. *of AAAI 2002 Spring Symposium (Sketch Understanding Workshop). Stanford, CA*, pages 86–90, 2002.

[50] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.*, 2001.

[51] Steffen Lauritzen and Frank Jensen. Stable local computation with conditional gaussian distributions. *Statistics and Computing*, pages 11:191–203, 2001.

[52] Steffen L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, pages v87, 1098–108, 1992.

[53] James V. Mahoney and Markus P. J. Fromherz. Handling ambiguity in constraint-based recognition of stick figure sketches.

[54] James V. Mahoney and Markus P. J. Fromherz. Interpreting sloppy stick figures by graph rectification and constraint-based matching. *Fourth IAPR International Workshop on Graphics Recognition, Kingston, Ontario, Canada*, 2001.

[55] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. *AAAI Spring Symposium: Sketch Understanding 2002*, 2002.

[56] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. *AAAI Sketch Understanding Symposium*, 2002.

[57] Ashutosh Malaviya and Reinhard Klette. A fuzzy syntactic method for on-line handwriting recognition. In *SSPR*, pages 381–392, 1996.

[58] Csar Pimentel Manuel J. Fonseca and Joaquim A. Jorge. Cali: An online scribble recognizer for calligraphic interfaces. *AAAI Spring Symposium: Sketch Understanding, March 25-27, Stanford CA*, 2002.

[59] Kevin Murphy. Dynamic bayesian networks. *Chapter in Probabilistic Graphical Models by Michael Jordan*, 2002.

[60] Kevin Murphy. Dynamic bayesian networks: Representation, inference and learning. *PhD Thesis, UC Berkeley*, 2002.

[61] Kevin Murphy and Mark Paskin. Linear time inference in hierarchical HMMs. *NIPS-01*, 2001.

[62] M. Muzumdar. ICEMENDR: Intelligent capture environment for mechanical engineering drawing. Master's thesis, Massachusetts Institute of Technology, 1999.

[63] Micheal Oltmans. Understanding naturally conveyed explanations of device behavior. Master's thesis, Massachusetts Institute of Technology, 2000.

[64] D. Pugh. Designing solid objects using interactive sketch interpretation. *Computer Graphics*, 1992.

[65] Plamondon R. and Srihari S. N. Online and offline handwriting recognition: A comprehensive survey. *PAMI, vol 22, no 1, Jan 2001*, 2001.

[66] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE, Vol. 77, no 2. 1989*, February 1989.

[67] SM Reynolds and JA Bilmes. Part-of-speech tagging using virtual evidence and negative training. *Proceedings of HLC/EMNLP*, 2005.

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

[69] P. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988.

[70] Tevfik Metin Sezgin. Generating domain specific sketch recognizers from object descriptions. *Student Oxygen Workshop, Gloucester MA*, July 17 2002.

[71] Tevfik Metin Sezgin and Randall Davis. HMM-based efficient sketch recognition. *Proceedings of IUI-2005*, 2005.

[72] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. *Proceedings of PUI-2001*, 2001.

[73] Michael Shilman, Hanna Pasula, Stuart Russel, and Richard Newton. Statistical visual language models for ink parsing. *AAAI Spring Symposium: Sketch Understanding, March 25-7, Stanford CA 2002*, 2002.

[74] Michael Shilman, Hanna Pasula, Stuart Russel, and Richard Newton. Statistical visual language models for ink parsing. *AAAI Sketch Understanding Symposium, Stanford CA 2002*, 2002.

[75] Michael Shilman and Paul Viola. Spatial recognition and grouping of text and graphics. *Eurographics 2004*, 2004.

[76] Saul Simhon and Gregory Dudek. Sketch interpretation and refinement using statistical models. *Eurographics 2004*, 2004.

[77] K. Tombre. Analysis of engineering drawings. In *GREC 2nd international workshop*, pages 257–264, 1997.

[78] Barbara Gans Tversky. What does drawing reveal about thinking? *Invited talk at First International Workshop on Visual and Spatial Reasoning in Design, Cambridge, MA.*, 1999.

[79] R.G. Uhl and N.d.V. Lobo. A framework for recognizing a facial image from a police sketch. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 586–593, 1996.

[80] David G. Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.

[81] Peter van Sommers. Drawing and cognition. descriptive and experimental studies of graphic production processes. *Cambridge University Press*, 1984.

[82] Olya Veselova. Perceptually based learning of shape descriptions. *Masters Thesis, Cambridge, MIT*, 2003.

[83] Toshinori Watanabe, Ken Sugawara, and Hiroshi Sugihara. A new pattern representation scheme using data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24(5): 579-590*, 2002.

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

[85] Liu Wenyin. Example-driven graphics recognition. *Structural, Syntactic, and Statistical Pattern Recognition, LNCS 2396*, 2002.

[86] R. Zeleznik. Sketch: An interface for sketching 3d scenes. In *Proceedings of SIGGRAPH'96*, pages 163–170, 1996.