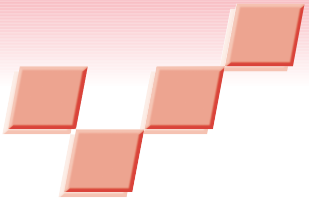


Sketch Interpretation Using Multiscale Models of Temporal Patterns

Tevfik Metin Sezgin and Randall Davis
Massachusetts Institute of Technology

The growing popularity of tablet PCs and intelligent pen-based interfaces have increased the importance of freehand sketch recognition algorithms as enabling technology. A recognition framework based on multiscale statistical models of temporal patterns significantly increases correct recognition rates, with no added computational penalties.

Sketching is a natural input modality that has received increased interest in the computer graphics and human–computer interaction communities. The emergence of hardware such as tablet PCs and handheld PDAs provides easy means for capturing pen input. These devices combine a display, pen tracker, and computing device, making it possible to capture and process sketches online, as they are drawn. Online recognition has two main advantages. First, it enables interpreting and displaying pen input as it is entered—for example, an engineer drawing a circuit diagram receives system feedback by observing the ink changing color in real time, indicating which circuit components are recognized. Second, an online sketching system provides access to stroke-ordering information as well as the end product of the drawing process. When we refer to temporal patterns in this article, we mean this stroke-ordering information.

In certain domains, temporal stroke orderings used when sketching objects contain predictable patterns that a system can use for object recognition.[1] We call these stroke-level patterns because they capture the probability of seeing a sequence of strokes with certain properties. For example, when people draw stick figures, one frequently seen stroke-level pattern is a sequence of a circular stroke, a vertical line, and two pairs of positively and negatively sloped lines, corresponding to the figure's head, body, arms, and legs.

Another temporal pattern in online sketches is an object-level pattern, which captures the probability of seeing a certain sequence of objects being drawn. Consider the domain of Unified Modeling Language class diagrams, drawn by software designers using rectangles to indicate classes and various arrows to indicate relations among classes, such as inheritance, generalization, and association. In this domain, when a designer draws a new class (indicated by a rectangle), it's natural to expect that the new object will soon be connected with an arrow to one or more of the objects drawn earlier. We describe this as an object-level pattern, indicating that drawing a class is followed by drawing some variety of arrow. The kind of arrow to expect can depend on the kind of class drawn. A final class, for example, can't be extended, which limits the kind of arrow that we can expect next.

In domains like the Unified Modeling Language, which has a graphical grammar of sorts, it's plausible to imagine writing down the grammar and using this to guide a UML-diagram recognition system. But few domains have patterns that are as well understood as those 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 that various objects can combine. A better way of incorporating object-level temporal patterns in a recognition framework would be to learn them, along with stroke-level patterns, from data.

In this article, we present our sketch-recognition framework, which uses data to automatically learn the object orderings that commonly occur when people sketch and then use the orderings for sketch recognition. The key features that make this framework novel include learning object-level patterns from data, handling objects comprising multiple strokes (*multistroke objects*) and objects that share strokes (*multiobject strokes*), and supporting continuous observable features. We also present an efficient graphical model implementation of our approach and report that a specialized inference algorithm known as the Lauritzen-Jensen stable conditional Gaussian belief propagation

 Published by the IEEE Computer Society 0272-1716/07/$20.00 © 2007 IEEE

should be used to avoid numerical instabilities in recognition. We also report results from experiments that help us establish a baseline for the recognition level achievable using temporal information alone. Moreover, we offer temporal information as a potential source of support to shape-based sketch recognizers.

Problem description

Just as approaches to sketch recognition vary, terminology as well as the problem definition itself also differ.

Terminology

By a sketch, we mean messy, informal, freehand drawings. Specifically, we are interested in recognizing sketches that use a fixed graphical vocabulary of symbols that, in turn, can be represented with structural descriptions.[1-4] Figure 1, for example, uses the standard vocabulary of graphical symbols from the electronic circuits domain. The graphical symbols that make up the domain are called objects (for example, resistors and transistors). Objects that comprise multiple strokes are called *multistroke objects*. Strokes that span multiple objects are called *multiobject strokes*.

We formally define a sketch $\mathcal{S} = S_1, S_2, \ldots S_N$ as a sequence of strokes captured using a digitizer.[1] A stroke S_i is a set of time-stamped points sampled between pen-down and pen-up events during sketching. Strokes are collected with an ink-collection application using the Tablet PC SDK, running on an Acer C110 with a 10.4-inch screen at 1,024×768 resolution.

Each stroke is preprocessed using the toolkit described in Sezgin et al.,[5] converting the sketch $S_{1:N}$ to a time-ordered sequence of geometric primitives (line and arc segments) $P_{1:T}$. (We use the Matlab notation begin:end, for example, 1:4 = 1, 2, 3, 4.) As is the case for edge detection in computer vision, the purpose of primitive extraction is to obtain a representation that's more expressive and manageable than a collection of pixels or points. Because a single stroke may result in more than one primitive, the total number of primitives is usually larger than the number of strokes.

We define *sketch recognition* as the segmentation and classification of a sketch. *Segmentation* is the task of grouping together primitives constituting the same object. *Classification* is the task of determining which object each group of primitives represents, such as a stick figure or a rectangle. Segmentation produces K groups $G = G_1, G_2, \ldots G_K$, and classification gives us the labels for the groups $L = L_1, L_2, \ldots L_K$. 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 supports multiobject strokes by allowing a stroke to be part of multiple objects (for example, drawing a box and an arrow or a resistor and two wires in a single stroke).

We refer to the sequence of features $\mathcal{O} = O_1, O_2, \ldots, O_T$ obtained from the primitives as *observations*. $\boldsymbol{O}_{G_i}$ is the sequence of observations corresponding to a group of primitives G_i, and $L(G_i)$ is the class label assigned to that group. Finally, $\lambda_{L(G_i)}$ represents the stroke-level temporal model corresponding to the label $L(G_i)$, and λ_{obj} refers to our model for object-level temporal patterns.

Problem formulation

Given a sequence of time-ordered primitives obtained from a sketch, our goal is to find a segmentation and classification of the primitives that maximizes the joint likelihood of stroke-level patterns and object-level patterns.

We formulate the problem starting with the stroke-level model. If all we had were the stroke-level patterns, then the recognition task would require finding a segmentation and classification of the primitives that maximizes the likelihood of the stroke-level patterns. In other words, over all possible ways in which we can group the primitives, $\mathfrak{G}$, and all possible ways in which we can label these groups, $\mathfrak{L}(G)$, the stroke-level model aims to find the grouping $G \in \mathfrak{G}$ and the labeling $L \in \mathfrak{L}(G)$ that maximizes the joint 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 ($\lambda_{L(G_i)}$). This amounts to finding the solution that maximizes the likelihood expression:

$$\underset{G \in \mathfrak{G}, L \in \mathfrak{L}(G)}{\operatorname{argmax}} \prod_{i=1}^{|L|} P\left(\mathrm{O}_{G_i} | \lambda_{L(G_i)}\right) \quad (1)$$

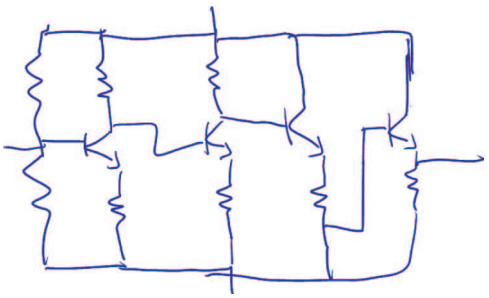
To obtain the expression corresponding to the stroke-level and object-level patterns, we multiply Equation 1 by a likelihood term corresponding to the likelihood of the label sequence $L_1, L_2, \ldots, L_K$, given our model for object-level patterns (λ_{obj}). This gives us the following expression, the solution to which is the segmentation and labeling of the input sketch:

$$\underset{G \in \mathfrak{G}, L \in \mathfrak{L}(G)}{\operatorname{argmax}} P\left(L_1, \ldots, L_K | \lambda_{\text{obj}}\right) \prod_{i=1}^{K} P\left(\mathrm{O}_{G_i} | \lambda_{L(G_i)}\right) \quad (2)$$

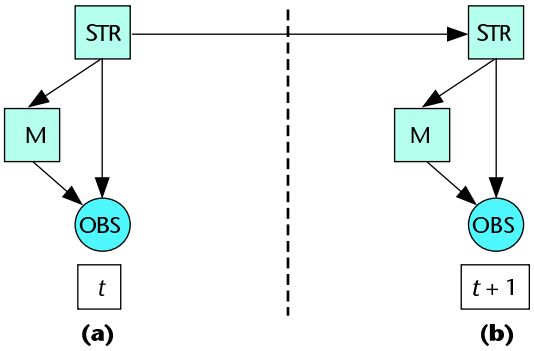
Approach

We can maximize the expression in Equation 2 over the set of all groupings and their labelings ($\mathfrak{G}$ and $\mathfrak{L}(G)$). As the vision- and sketch-recognition literature documents, exhaustive search of this space is computationally intractable.

Because we're modeling sequential patterns, we assume that in our target domains both stroke- and object-level patterns can be modeled as products of first-



1 A hand-drawn circuit diagram.



2 The dynamic Bayesian network representing the model for capturing stroke-level patterns: (a) initial frame and (b) repeating frame. This fragment has a dual representation as a hidden Markov model with continuous observations and $|M|$ mixtures.

order Markov processes. This will let us efficiently compute the maximum likelihood estimates to learn the parameters of our stroke- and object-level models, and it will enable efficient recognition. Our definition of efficiency encompasses the CPU time spent when our algorithm runs on realistic problems as well as the theoretical bounds on its algorithmic complexity. As we'll discuss in a moment, our algorithm takes a few seconds to interpret moderate-sized sketches, and the time complexity of inference in our dynamic Bayesian network-based representation scales linearly with respect to the problem size.

The model

We represent our model of temporal patterns using DBNs. Bayesian networks encode the joint probability of a set of variables $Z = \{Z_1, \ldots, Z_n\}$, where the network's graphical structure encodes the conditional dependencies among the variables. DBNs extend Bayesian networks that model the joint distribution of a set of variables over time by representing the conditional dependencies between the variables using a pair of Bayesian networks $(B_1, B_{\rightarrow})$. The network 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. Bayesian networks and DBNs have visual representations in the form of graphs, hence they're called graphical models. The rest of our discussion assumes you are familiar with DBNs. (For a more thorough discussion and a pointer to an excellent review, see Murphy and Paskin's work.)[6]

The input. The input to our model is the observation sequence $\mathcal{O} = O_{1:T}$, obtained by computing features from corresponding primitives $\mathcal{P} = P_{1:T}$. Working with primitives, rather than strokes, lets us handle multiobject strokes, which is an essential feature in certain domains but is sometimes unsupported for reasons of efficiency or architectural limitations.[2] By breaking strokes into primitives and having a model that allows primitives obtained from a single stroke to be assigned to different objects, we allow multiple objects to share a stroke (but not a primitive).

In sketch recognition, we deal with data that is most naturally described using geometric features such as the shape of a stroke segment, the length and orientation of line segments, and the radii of circles. Some of these features are categorical—for example, a stroke segment's shape can be arc, line, and so on—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, we support both discrete and real-valued (continuous) observations. In addition to letting us have a richer set of features, supporting real-valued features makes it possible to avoid discretization issues, such as choosing the optimal discretization parameters (for example, number of bins and bin sizes).

The stroke-level model. For the sake of clarity of presentation, we present our model bottom-up, starting with the stroke-level model. For each object class, we need a stroke-level model. The stroke-level model computes 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 like What is the likelihood of seeing the observation sequence $\boldsymbol{O}_{G_i}$ obtained from a group of primitives G_i under the rectangle model, or more formally, What is $P(\boldsymbol{O}_{G_i} \mid \lambda_{\text{rectangle}})$? This question corresponds to the likelihood term $P(\boldsymbol{O}_{G_i} \mid \lambda_{L(G_i)})$ in Equation 1 and the innermost term in Equation 2.

Figure 2 shows the DBN fragment corresponding to our stroke-level model. The observable node OBS at time t represents the features extracted from primitive P_t. As we noted, we encode the stroke-level patterns using a first-order Markov process. The discrete node STR in Figure 2 captures the dynamics of this Markov process. The M variable is a discrete mixture variable and lets us 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 as products of a Markov process.

As a simple example, consider the hypothetical scenario in Figure 3. The user draws NPN transistors using exactly two orderings, each appearing equally frequently. In this case, the stroke-level model with single mixtures ($|M| = 1$) might learn that starting in the initial state, with probability $p = 1.0$, we observe a vertical line. Then, we either move to a new state with $p = 0.5$ and observe a positively sloped line, or, with $p = 0.5$, we move to another state and observe a negatively sloped line. In this respect, the stroke-level model acts exactly like a regular hidden Markov model (HMM). During training, we estimate parameters of the STR node on the basis of the patterns in the training data provided through the OBS variable. The training process can adjust the model parameters for a given training data in many ways, and we rarely learn models that exactly match our intuition about the underlying drawing orders, as was the case in this example.

The combined model. Our combined model can be described as a DBN or, in terms of its dual represen-

tation, a hierarchical HMM with continuous observables modeled using mixtures of Gaussians. We describe it as a DBN because DBNs generalize HHMMs, and the DBN representation is more efficient.

We create the combined model by augmenting the stroke-level model with two nodes, OBJ_t and END_t (see Figure 4). The OBJ_t node is a multivalued, discrete variable that represents our belief about which object primitive P_t is a part of. Its cardinality is the same as the number of classes we can recognize. The value of END_t indicates our belief that the user has just completed drawing object 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 $OBJ_{1:T}$, as well as hypotheses on where each object begins and ends ($END_{1:T}$). The joint probability over $OBJ_{1:T}$ corresponds to $P(L_1, \ldots, L_K|\lambda_{obj})$, the first term in Equation 2. Combined with the stroke-level likelihood term $P(\boldsymbol{O}_{G_i} | \lambda_{L(G_i)})$ from Equation 1, the combined model lets us compute the most probable values of $OBJ_{1:T}$ and $END_{1:T}$, which gives us the optimal segmentation and labeling of the input sketch.

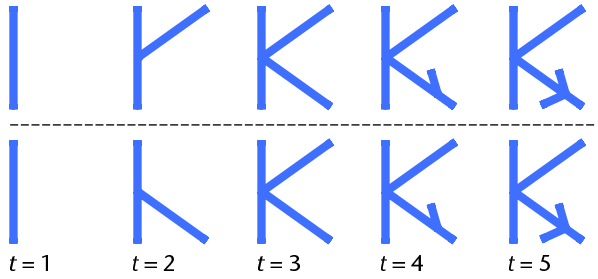
The OBJ node's value in each time slice is conditioned on the END and OBJ variable values from the previous slice. This encodes the observation that our belief about the value of OBJ_{t+1} is based on the values of OBJ_t and END_t. The justification for this dependence is that we would expect the OBJ node's value to change only if the user has just finished an object—that is, END_t = true—in which case the next object we would expect to see would depend on what object was just completed. The two new interslice arcs introduced into our model, $OBJ_t \rightarrow OBJ_{t+1}$ and $END_t \rightarrow OBJ_{t+1}$, cross only a single slice boundary, thus our model remains first-order Markovian, enabling efficient training and recognition.

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 $P(L_1, \ldots, L_K|\lambda_{obj})$, the first term in Equation 2. In the combined model, the observations' distribution is based on the OBJ node's value given by $P(OBS_t|OBJ_t, STR_t, M_t)$. The OBJ node determines the choice of which stroke-level process the system activates, because STR_{t+1} is conditioned on STR_t, END_t, and OBJ_{t+1}.

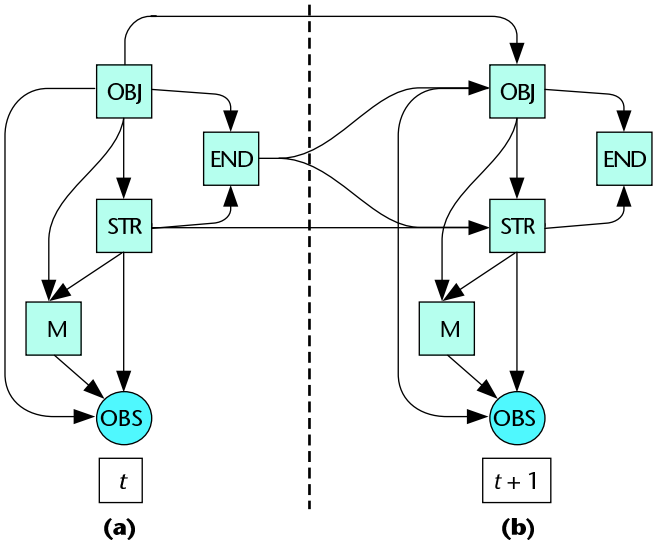
Implementation issues

One should address two issues in implementing the model we've described; both relate to the choice of which inference algorithm to use.

As we mentioned, our model has a dual representation as an HHMM. Unfortunately, the original inference algorithm for HHMMs is complicated and has $O(T^3)$ time complexity (where T is the length of the observation sequence).[6] In our case, T is the total number of primitives in a sketch, and this makes the model impractical for situations that require real-time feedback. By contrast, inference in the particular DBN that we use takes linear time, which in turn forms the basis of our system's efficiency.[6] It's therefore essential to use the DBN representation, or convert any HHMM-based representation to a DBN prior to inference and learning.



3 A hypothetical scenario. The user draws NPN transistors in only two ways.

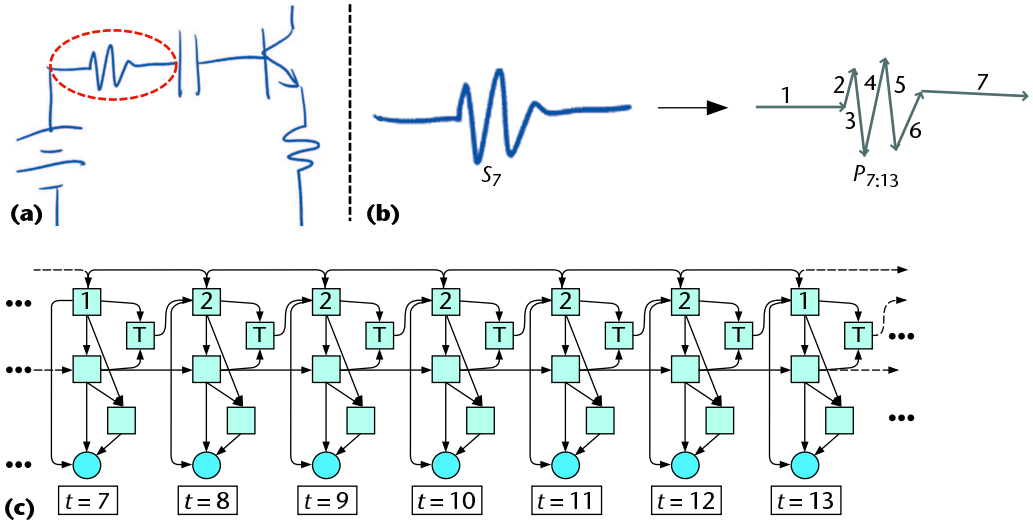


4 The dynamic Bayesian network representing our model.

A major issue that arose during implementation was the numerical instabilities that occur during training because of 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. Unfortunately, this algorithm is susceptible to numerical underflow. Although the machine learning literature has documented the numerical instability, the instability isn't well known because it occurs rarely in practice. The problem appears in the form of singular matrices during inference, and it is hard to pinpoint the source of error. To avoid this numerical instability, we must use a specialized algorithm known as the Lauritzen-Jensen stable conditional Gaussian belief propagation.[7]

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 $P(A|\text{Parents}(A))$ relating a node A and its parents is the same for each node A in B_1 and $B_{\rightarrow}$.[3]

During training, we know what class each primitive belongs to, and we know when objects begin and end, so



5 A Bayesian network is created by unfolding (B_1, $B_{\rightarrow}$) during recognition. (a) A circuit fragment. (b) A primitive extraction for stroke 7, which is also a multiobject stroke. (c) The corresponding fragment of the unfolded network. Expected values of $OBJ_{7:13}$ and $END_{7:13}$ are also shown. For $OBJ_{7:13}$, the values {1, 2} represent wire and resistor. For $END_{7:13}$, {T, F} represent true and false.

the values of OBJ and END are observable and are supplied during training. To summarize, for each labeled sketch, we use the values $OBJ_{1:T}$, $END_{1:T}$, and $OBS_{1:T}$ to estimate the model parameters.

Recognition. The input to recognition is $OBS_{1:T}$. During recognition, our system computes the values of $OBJ_{1:T}$ and $END_{1:T}$ that maximize the likelihood of the observations $OBS_{1:T}$.

We do this using the stable conditional Gaussian belief propagation algorithm. Interpretation of complete sketches comparable to that in Figure 1 takes less than 2 to 3 seconds on a 2-GHz Pentium 4 machine.

Results

To see whether modeling object-level patterns improves over modeling only stroke-level patterns, we tested our system using sketches from the electronic circuit diagrams domain. Although we found that many other domains contain predictable temporal patterns (including UML diagrams, finite-state diagrams, military course of action, and emoticons[1]), we chose to focus on the electronic circuit diagrams domain because it represents a group of domains that can be characterized as object–connector diagrams (such as organizational charts, flowcharts, and UML diagrams). This domain also illustrates our model's ability to deal with multiobject strokes (for example, stroke 7 in Figure 5b) and objects with varying numbers of components (for example, resistors with varying numbers of humps).

Data collection

We collected circuit diagrams from undergraduate students who had recently taken the Microelectronic Circuits and Devices course at the Massachusetts Institute of Technology and were familiar with the textbook used in the class. A total of eight participants contributed circuit diagrams. All except participant 7 were right-handed. We asked the participants to draw circuits selected from their course textbook using a sufficiently expressive set of electronic circuit components. (The components were NPN transistors, resistors, capacitors, batteries, and wires, so |OBJ| = 5.)

During data collection, we first showed participants diagrams of circuits that we wanted them to draw. We gave each participant time to study the circuits until he or she understood how they worked, then we asked the participants to explain the circuits verbally, to confirm their understanding.

We next removed the textbook diagram and asked them to draw the circuit using a tablet PC, allowing them to consult the original circuit diagram if needed. Figure 1 is an example of a participant's drawing.

To train our model, we created labeled training data by manually annotating 10 sketches per participant.

Generating the observation sequence

Both training and classification require converting a sketch to an observation sequence $O_{1:T}$. Using an early sketch-processing toolkit,[5] we first preprocess each sketch to obtain a sequence of primitives $P_{1:T}$ (in our case simply using line segments is sufficient because our domain has no objects with curved strokes). Note that the primitive extraction step is prone to error and does not always produce what a human would expect—that is, it might miss corners or break straight-looking lines unnecessarily. For each primitive P_i, we obtain an observation vector $\mathbf{O}_t$ represented as a five-tuple $(l_t, \Delta l_t, \theta_t, \Delta\theta_t, sgn_t)$ where

- l_t is the length of P_i,
- Δl_t is relative length (l_t/l_{t-1},1 for $t = 1$),
- θ_t is the angle with respect to the horizontal axis,
- $\Delta\theta_t$ is the measure of relative angle between P_i and P_{i-1}, and
- sgn_t is the direction that the stroke turns when moving from P_{i-1} to P_i.

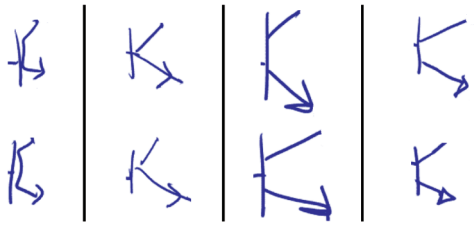
The value of $\Delta\theta_t$ is given by the magnitude of the cross product $\mathbf{u}\times\mathbf{v}$ of vectors $\mathbf{u}$, $\mathbf{v}$, which are length-normalized versions of P_i and P_{i-1} pointing in the direction of pen movement along each primitive. The turn direction (sgn_t) is the only discrete feature and is set to 0 for negative values of $\mathbf{u}\times\mathbf{v}$ and 1 for positive values or $t=1$. The use of absolute length is based on our observation that for a given device and user, absolute length is a reliable feature with high discriminative power. Furthermore, once the system learns the corresponding Gaussian mixtures, the mixtures can easily be scaled to work with data collected using a different resolution device. The use of absolute angle is based on our observation that components in circuit diagrams mainly appear in two canonical orientations. We prefer learning a few canonical orientations (with the consequent limitations) rather than not being able to use this informative feature to gain orientation invariance. Finally, we prefer using stroke-based edge features. For future work, we plan to study how image-based features such as shape contexts or SIFT (scale invariant feature transform) features would fare.

Recognition results

As Figure 6 shows, drawing styles show significant variation across users. A naive attempt to build general models to use with new users trained on data from other users would necessarily result in less accurate results. Therefore, to get an accurate measurement of how much recognition is possible using temporal information alone, we trained personalized models for each user. (For details on approaches that don't use temporal features, see the "Related Work in Sketch Recognition" sidebar on page 36.) An interesting question is whether any subpopulations show a useful degree of uniformity so that we can use data from the subpopulation to train a model for a novel user. Proper treatment of this issue can be quite involved. The only attempt at exploring this idea so far has been in the area of low-level ink processing, which uses clustering methods for learning subgroups of users to build corner-detection methods adaptive to novel users.[8]

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 that modeled only stroke-level patterns (no object-level patterns). We obtained the baseline system by setting $P(\mathrm{OBJ}_{t+1}|\mathrm{OBJ}_t, \mathrm{END}_t)$ to be uniform ($1/|\mathrm{OBJ}|$) if END_t = true. For END_t = false, we set it to 1 for $\mathrm{OBJ}_{t+1}=\mathrm{OBJ}_t$, and 0 otherwise. The baseline has the same network topology and the same junction tree, hence, inference for the baseline method has the same computational complexity as our method. In both models, we used the values $|\mathrm{STR}|=6$ and $|M|=3$.

Quantitative results. Table 1 shows the average correct recognition rates for the baseline model and our multiscale model on sketches collected from the eight participants (μ_b, μ_m). As the table shows, the improvement in absolute terms ranges from 2.1 to 6.2 percent.



6 Examples of transistors drawn by different users show structural variation. Each column shows a different user's sketches. Note the difference in routes when drawing the emitter and the current direction.

The table also shows the average and maximum reduction values in the error rates in terms of percentages (Δ_{err} and maxΔ) for each user. On average, modeling object-level patterns always improves performance, allowing up to 65 percent of misrecognition errors to be corrected.

A paired t-test comparing the average correct recognition rates for each user finds the increase to be statistically significant for $p<0.05$ and 7 degrees of freedom. We believe the reported decrease in the error rates would substantially improve users' satisfaction with the recognition system in actual use.

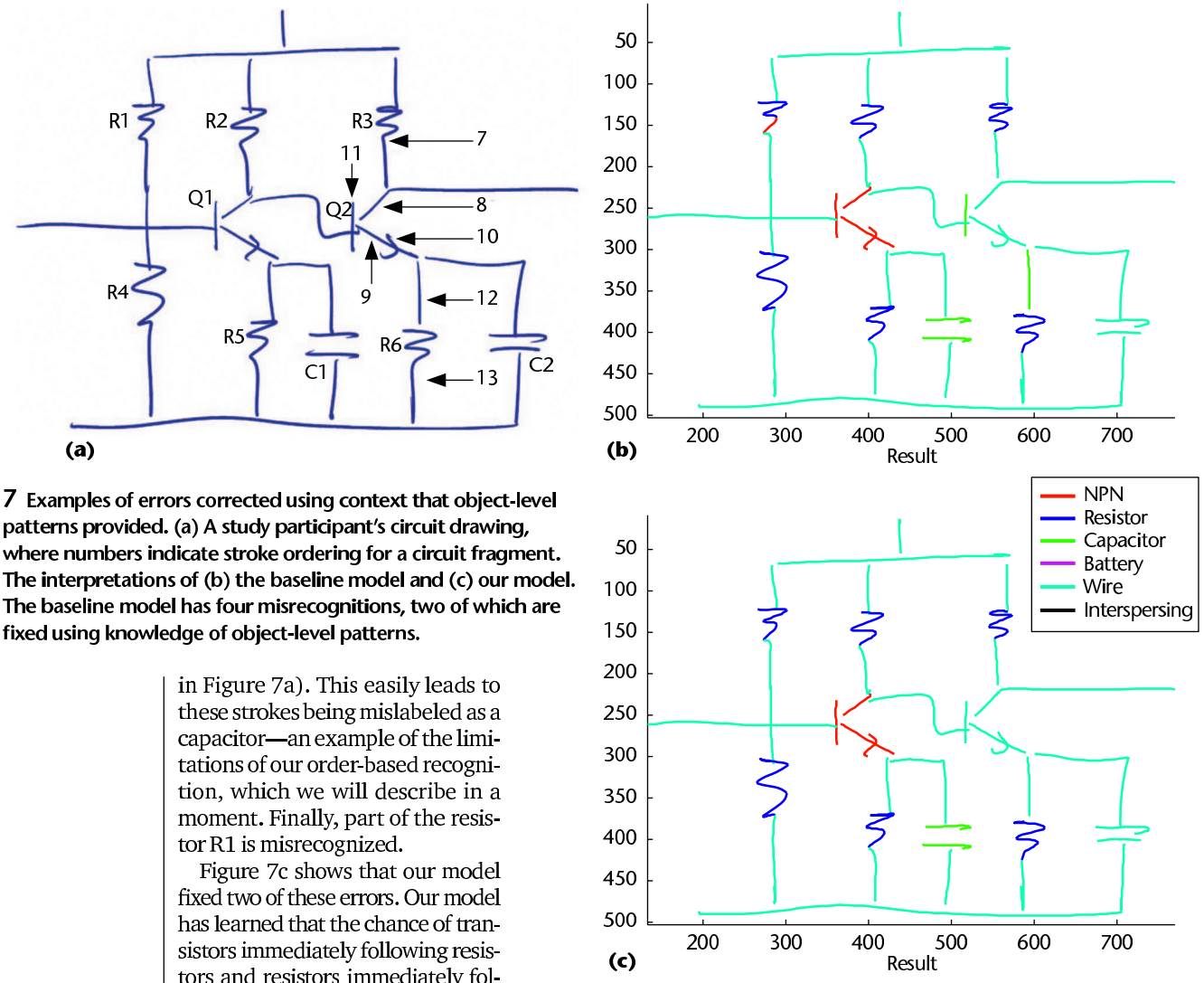
Note that the primitive extraction step is prone to error. We have not tested the sensitivity of our approach with respect to the errors made at the primitive extraction step. However, we didn't correct any of the errors that the primitive extraction step made. Both training and classification were subject to these errors, and our results reflect the errors made at the primitive extraction step.

Qualitative results. Exploring the kinds of errors avoided by modeling object-level patterns is instructive. Consider the amplifier circuit in Figure 7a (on the next page). The figure also shows the stroke ordering for the fragment of the circuit diagram that interests us. Figures 7b and 7c show the two interpretations of this circuit.

Figure 7b shows the interpretation obtained by running the baseline method, which encounters four recognition errors. The capacitor C2 is misrecognized as wires, most likely 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. 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

Table 1. Average correct recognition rates for two models.

	Participant ID							
	1	2	3	4	5	6	7	8
μ_b (baseline model)	83.2	86.5	85.2	73.8	87.1	90.7	79.1	85.0
μ_m (our multiscale model)	89.4	89.4	87.3	77.4	89.8	93.0	84.6	88.2
Δ_{err}	36.9	21.4	14.1	13.7	20.9	24.7	18.6	21.3
maxΔ	65.0	45	31.2	15.0	35.5	40.0	30.7	54.5



7 **Examples of errors corrected using context that object-level patterns provided. (a) A study participant's circuit drawing, where numbers indicate stroke ordering for a circuit fragment. The interpretations of (b) the baseline model and (c) our model. The baseline model has four misrecognitions, two of which are fixed using knowledge of object-level patterns.**

in Figure 7a). This easily leads to these strokes being mislabeled as a capacitor—an example of the limitations of our order-based recognition, which we will describe in a moment. Finally, part of the resistor R1 is misrecognized.

Figure 7c shows that our model fixed two of these errors. Our model has learned that the chance of transistors immediately following resistors and resistors immediately following capacitors, with no wires in between, is small. So, these interpretations are penalized. As a result, the system avoids the two misrecognitions.

Inspecting the conditional probability table $P(\text{OBJ}_{t+1}|\text{OBJ}_t, \text{END}_t)$ reveals the kinds of object-level patterns that our models learn. For the eight participants in our study, all the models learned (not surprisingly) that, most of the time, circuit components are preceded and followed by wires.

A more interesting observation comes from looking at what happens when a wire doesn't follow an object. In these cases, we see that components that frequently appear together in conceptually meaningful circuit fragments have a higher probability of being drawn consecutively. For example, resistors appear in series when drawing voltage dividers, hence when a resistor is not immediately followed by a wire temporally, it's more likely to be followed by a resistor than by anything else. It's conceivable that a circuit-design expert might predict the existence of such patterns and try to incorporate them in the design of a circuit-recognition system, but this would be a laborious task and require precision in balancing biases for different kinds of patterns. Our model learns and uses such patterns automatically in a mathematically sound framework.

Discussion

Our experiments also helped us identify a limitation of relying on temporal patterns alone. Our approach doesn't incorporate any spatial or geometric constraints beyond those used to encode stroke sequences. As a result, recognition is based strictly on temporal patterns and not on shapes. Therefore, our system can incorrectly classify a sequence of strokes that has the right temporal character but the wrong shape. For example, inspection of the circuit in Figure 7a reveals that strokes 11 and 12 were drawn one after the other, they are parallel, and they have roughly the same length. As a result the system misclassified them as capacitors, even though the strokes are offset from one another. We can argue, then, that augmenting the feature set to include relative positions of consecutive primitives will fix the error in this specific case. But, incorporating shape information requires modeling complex long-term dependencies (constraints) between primitives arbitrarily apart in time, such as those described by Alvarado and Davis.[2] This is beyond the reach of our first-order Markovian model.

Our evaluation also revealed a surprising drawing convention employed while drawing transistors.

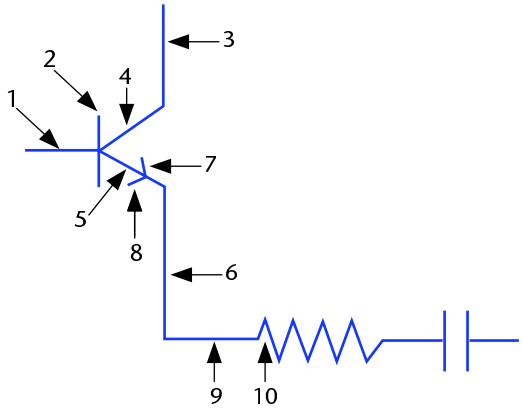
Although the participants completed each circuit component before starting a new one, four of them sometimes interspersed wires during the course of drawing transistors. By interspersing, we refer to the situation where the user starts drawing one object but draws one or more other objects before the first is completed (for example, wires 3 and 6 in Figure 8 on the next page). Because our model assumes the users complete each object before moving on to the next, interspersed drawing causes misrecognitions.

In our case, interspersings happened only while drawing transistors.[6] To measure the effects of interspersing on correct recognition rates, we selected a control group of 10 circuits containing two to four transistors with no interspersings and a test group that contained versions of these sketches with interspersings. We obtained the interspersed version of each sketch by moving one of the wires preceding or following a transistor backward or forward in time. The way we set up our control and test sets ensures that we measure the misrecognition effects due to interspersing only.

Table 2 summarizes the recognition results obtained for the control and test sets carried out in a leave-one-out cross-validation setup. In both cases, the training used circuits with no interspersings, consistent with our model's assumptions. As the results in Table 2 indicate, violation of our "one object at a time" assumption always introduces new recognition errors. Even when the number of misclassifications is normalized by the number of interspersings, a transistor-wire interspersing causes 1-4 incorrect classifications per interspersing. In the "Future work" section, we explain how to address this issue by modeling interspersed drawing behavior explicitly.

Future work

A promising direction to explore would be developing and evaluating a complete pen-based interface to a computer-aided circuit design tool, using our system as the recognition engine. This would let us quantify the subjective utility of recognition and evaluate user reactions in a real design setting. Issues to investigate include the editing method (modeless versus modal) and the display of the recognition results (delayed versus immediate feedback) and their effect on the drawing behavior (for example, Do users still intersperse objects?). Similar experiments can be carried out for other domains, such as UML diagrams, Web page design, and mechanical engineering.



8 Interspersed drawing causes misrecognitions. Over the course of drawing the transistor, the user draws two other objects (wires 3 and 6). Numbers indicate drawing order.

We are also developing ways to modify our model so that it handles the interspersed drawing issue. The idea is to make the fact that users can intersperse certain types of objects an explicit part of the DBN model. We can do this by extending the state space of each time slice with a new variable that tracks whether the user is interspersing any objects, and, if so, what objects are being interspersed. Early results show that it's possible to extend the state space of our model using DBNs' parsimonious representation property and obtain a model where both training and inference remains tractable.[9]

The motivation for our work was to identify the kinds of temporal patterns present in online sketches and to present an approach that can perform recognition based on such patterns. We have illustrated that sketches contain a fair amount of rich temporal patterns that can be used for recognition. However, we don't expect temporal patterns to be used alone. Ultimately, we envision a unified recognition architecture that combines the efficient segmentation and recognition features of temporal approaches with the rich constraints and representational power of shape-based models.

Our choice of modeling the sketching process as a first-order Markov process doesn't allow us to exploit spatial or geometric constraints between arbitrary components of a shape. A naive attempt to incorporate such constraints using higher-order Markov processes is

Table 2. Misrecognized primitives per interspersing.

Parameters	Sketch ID 1	2	3	4	5	6	7	8	9	10	μ
Total primitives in each sketch	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 in the control set	5	5	2	1	4	2	5	8	9	16	5.7
Missed in the test set	7	9	15	12	12	12	13	13	11	19	12.3
Absolute difference*	2	4	13	11	8	10	8	5	2	3	6.6
Primitives missed per interspersing	1	2	3.25	2.75	2.67	3.33	4	2.5	1	0.75	2.33

* The number of primitives that our system misses because it cannot handle interspersings.

Related Work in Sketch Recognition

A large body of work in sketch recognition discusses approaches that don't use temporal features. Most of these approaches, which complement ours, use algorithms based on structural and syntactic methods. Structural pattern recognition methods (also known as model- or template-based methods) formulate the recognition problem on the basis of objects' implicit or explicit representations in terms of their structure (such as components, subcomponents, and their relationships). Examples include the work of Alvarado and Davis[1] and of Shilman et al.[2] The main drawback of such methods is their high computational requirements that make them unsuitable for real-time recognition of realistic sketches. These models also require experts to specify domain knowledge or manually input structural descriptions of the objects in the domain. Further complicating this requirement is the fact that not everyone draws objects the same. For example, Figure 6 in the text shows how transistors drawn by four participants were structurally different. Thus, in a structural-recognition setting, the expert might have to create different object descriptions for different users. Our system learns each variation from examples.

Other nontemporal recognition algorithms include Gennari et al.'s work, which requires an expert to enter domain information.[3] Our system doesn't require expert knowledge. Shilman and Viola describe a spatial recognition and grouping method for text and graphics.[4] They make some assumptions about the objects in the domain to keep the search tractable (for example, objects in their domain have no more than eight strokes or the strokes constituting an object are within a certain distance). These assumptions seem to work for flowchart-like drawings, where the connectors and objects are sufficiently separated from one another. However, the practicality of these algorithms has yet to be demonstrated in general for domains where objects vary in size and shape or where assumptions on the object size and scale might not hold.

The handwriting recognition community has come up with numerous methods for recognizing single-stroke or segmented-pen input using hidden Markov models (HMMs). These systems take advantage of a vast number of robust preprocessing and segmentation methods to simplify the problem to an isolated recognition problem.

Several systems have used HMM or hierarchical HMM-based frameworks to recognize pen input. In the sketch recognition community, Anderson, Bailey, and Skubic's work describes an HMM-based symbol recognizer that uses chain-code-like features to recognize isolated symbols.[5] Our work doesn't assume segmented input and builds a joint model for complete sketches.

Our previous work describes an HMM-based sketch recognition system that does segmentation and recognition by combining outputs of individual HMMs using dynamic programming.[6] This method is a weaker version of our baseline method in that it doesn't support continuous features. Its two-layer HMM and dynamic-programming recognition method suffers from the $O(T^3)$ complexity, just like hierarchical HMMs.

Simhon and Dudek present a sketch interpretation and curve refinement system that uses an HHMM.[7] This work and ours share the same motivation. 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 HHMMs' $O(T^3)$ complexity. We also support multistroke objects, and our use of real-valued continuous observations allow using a rich set of features.

References

1. C. Alvarado and R. Davis, "Sketchread: A Multi-Domain Sketch Recognition Engine," *Proc. 17th ACM Symp. User Interface Software and Technology* (UIST 04), ACM Press, 2004, pp. 23-32.
2. M. Shilman et al., "Statistical Visual Language Models for Ink Parsing," *AAAI Spring Symp.: Sketch Understanding*, AAAI Press, 2002, pp. 126-132.
3. L. Gennari, L.B. Kara, and T.F. Stahovich, "Combining Geometry and Domain Knowledge to Interpret Hand-Drawn Diagrams," *AAAI Fall Symp.*, AAAI Press, 2004, pp. 64-70.
4. M. Shilman and P. Viola, "Spatial Recognition and Grouping of Text and Graphics," *Proc. Eurographics Workshop Sketch-Based Interfaces and Modeling*, Eurographics Assoc., 2004, pp. 91-95.
5. D. Anderson, C. Bailey, and M. Skubic, "Hidden Markov Model Symbol Recognition for Sketch-Based Interfaces," *AAAI Fall Symp.: Making Pen-Based Interaction Intelligent and Natural*, AAAI Press, 2004, pp. 15-21.
6. T.M. Sezgin and R. Davis, "HMM-Based Efficient Sketch Recognition," *Proc. 10th Int'l Conf. Intelligent User Interfaces* (IUI 05), ACM Press, 2005, pp. 281-283.
7. S. Simhon and G. Dudek, "Sketch Interpretation and Refinement Using Statistical Models," *Proc. 15th Eurographics Symp. Rendering* (EGSR 04), Eurographics Assoc., 2004, pp. 23-32.

impractical because our model's state space increases exponentially with the process order. A better approach would be to explore ways of incorporating shape information by connecting external recognizers that use spatial or structural information into our model using a mechanism known as *virtual evidence nodes* in the graphical models machine-learning community. Another approach would be to learn a composite function of spatial and temporal information using a dynamic programming framework.[9] This work would be a first step toward combining temporal and spatial/structural information for sketch recognition. ■

Acknowledgments

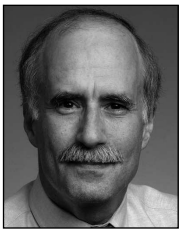
This work was supported in part by the MIT/Microsoft iCampus Project and by a grant from the Intel Corp. We thank the anonymous referees for valuable feedback that improved our article's presentation and content.

References

1. T.M. Sezgin and R. Davis, "HMM-Based Efficient Sketch Recognition," *Proc. 10th Int'l Conf. Intelligent User Interfaces* (IUI 05), ACM Press, 2005, pp. 281-283.

2. C. Alvarado and R. Davis, "Sketchread: A Multi-Domain Sketch Recognition Engine," *Proc. 17th ACM Symp. User Interface Software and Technology* (UIST 04), ACM Press, 2004, pp. 23-32.
3. M. Shilman et al., "Statistical Visual Language Models for Ink Parsing," *AAAI Spring Symp.: Sketch Understanding*, AAAI Press, 2002, pp. 126-132.
4. L. Gennari, L.B. Kara, and T.F. Stahovich, "Combining Geometry and Domain Knowledge to Interpret Hand-Drawn Diagrams," *AAAI Fall Symp.*, AAAI Press, 2004, pp. 64-70.
5. T.M. Sezgin, T. Stahovich, and R. Davis, "Sketch Based Interfaces: Early Processing for Sketch Understanding," *Proc. 2001 Workshop Perceptive User Interfaces* (PUI 01), ACM Press, 2001, pp. 1-8.
6. K. Murphy and M. Paskin, "Linear Time Inference in Hierarchical HMMs," *Advances in Neural Information Processing Systems*, T.G. Dietterich, S. Becker, and Z. Ghahramani, eds., MIT Press, 2001.
7. S. Lauritzen and F. Jensen, "Stable Local Computation with Conditional Gaussian Distributions," *Statistics and Computing*, vol. 11, no. 2, 2001, pp. 191-203.
8. S. J. Cates, "Using Context to Resolve Ambiguity in Sketch Understanding," master's thesis, Massachusetts Inst. of Technology, Dept. Electrical Eng. and Computer Science, 2006.
9. T.M. Sezgin, "Online Sketch Recognition from a Dynamic Perspective," doctoral dissertation, Massachusetts Inst. of Technology, Dept. Electrical Eng. and Computer Science, June 2006.

Tevfik Metin Sezgin *is a postdoctoral researcher at the University of Cambridge Computer Laboratory. His research interests include intelligent human–computer interfaces, multimodal sensor fusion, and HCI applications of machine learning. He has a PhD from the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory. Contact him at mtsezgin@csail.mit.edu.*



Randall Davis *is a full professor in the Computer Science and Engineering Department and a research director at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory. His research interests include developing tools that permit natural multimodal interaction with computers by creating software that understands users as they sketch, gesture, and talk. He has a PhD from Stanford University. Contact him at davis@csail.mit.edu.*

For further information on this or any other computing topic, please visit our Digital Library at http://www.computer.org/publications/dlib.

Get access

to individual IEEE Computer Society documents online.

More than 100,000 articles and conference papers available!

$9US per article for members

$19US for nonmembers

www.computer.org/publications/dlib

IEEE computer society