EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

Temporal Sketch Recognition in Interspersed Drawings

Tevfik Metin Sezgin [1] and Randall Davis [2]

[1]University of Cambridge, Computer Laboratory, Cambridge, UK
[2]Massachusetts Institute of Technology, CSAIL, Cambridge, MA, USA

Abstract

Sketch recognition has been recognized as an enabling technology for pen-based interfaces. Previous work in the field has shown that in certain domains the stroke orderings used when drawing objects contain temporal patterns that can aid recognition. So far, systems that use temporal information for recognition have assumed that objects are drawn one at a time. This paper shows how this assumption can be relaxed to permit temporal interspersing of strokes from different objects. We describe a statistical framework based on Dynamic Bayesian Networks that explicitly models the fact that objects can be drawn interspersed. We present recognition results for hand-drawn electronic circuit diagrams. The results show that handling interspersed drawing provides a significant increase in accuracy.

Categories and Subject Descriptors (according to ACM CCS): I.5.4 [Pattern Recognition]: Applications

1. Introduction

Sketching is typically (and unconsciously) rather stylized in the sense that people have routine habits in the way they sketch. For example, people typically draw enclosing objects first and use a left-to-right stroke ordering when drawing symmetric objects. There is psychological evidence attributing such ordering phenomenon to motor convenience, part salience, hierarchy, geometric constraints, planning and anchoring [Tve99, vS84].

Existence of ordering patterns during drawing is significant from a recognition perspective because as has previously been demonstrated in a variety of domains, temporal stroke orderings can be used to aid recognition [SD04, ABS04, SD05, SD07]. All these systems, however, make certain assumptions that limit the complexity of the inputs they can accommodate. For example, the simplest approach assumes the scene contains only one object, drawn in a single stroke [ABS04]. Other systems allow recognition in scenes with multiple objects with the restriction that objects or complete object components are drawn using a single stroke [SD04]. Another approach allows scenes with multiple objects and objects consisting of multiple strokes but assumes that no objects share strokes [SD05]. A more recent framework allows stroke sharing between shapes under certain conditions and shows how common stroke orderings as well as object orderings can be used for recognition [SD07].

Even so, one key assumption that all these systems make about free-hand drawing — one that has received much criticism — is the assumption that people complete each object before moving on to draw the next one. Real-world data shows this to be untrue. For example, our analysis of free-hand analog electronic circuit diagrams collected from electrical engineers shows that there are indeed cases where people start drawing a new object before they complete the current one. This drawing behavior, which we call *interspersed drawing*, occurs in other domains as well [AL07], and the ability to deal with interspersed drawing is recognized as a major task that sketch recognizers should support [AL07]. This paper is focused on this issue, and shows how stroke ordering information can be used for sketch recognition in presence of interspersed drawing. Additional key features of our recognition framework include its ability to learn various kinds of temporal patterns from data, the ability to handle multi-stroke objects and multi-object strokes, and support for continuous observable features.

Next, we formally define the sketch recognition task and describe the interspersed drawing phenomena. In section 3, we describe a recognition framework based on Dynamic Bayesian Networks (DBNs) that models online sketching as a stochastic process employing specialized constructs called switching parents. Section 4 reports evaluation results showing that a significant percentage of misrecognitions can be

Sketch-Based Interfaces and Modeling 2007, Riverside, CA, August 02-03, 2007.
© 2007 ACM 978-1-59593-913-5/07/0008 $5.00

avoided by explicitly modeling interspersed drawing behavior. We conclude with a broader discussion of the related work and point out possible future directions.

2. Problem Definition

Informally, the goal of sketch recognition is to break up digital ink drawn by user into smaller pieces and assign labels to groups that constitute objects in the domain. We focus here on the domain of hand-drawn electronic circuit diagrams, but our recognition algorithm is not specific to this domain. Objects in our domain are wires, resistors, capacitors, npn-transistors and batteries.

2.1. Terminology

We adopt the terminology and notation used in [SD07]. A ***sketch*** $\mathscr{S} = S_1, S_2, ...S_N$ is defined as a sequence of strokes captured using a digitizer, preserving the drawing order. A ***stroke*** is a set of time-ordered points sampled between pen-down and pen-up events. Each stroke is broken into geometric primitives (e.g., lines, arcs) called ***primitives*** as part of the preprocessing of the sketch.† Let $\mathscr{P} = P_{1:T} = P_1, P_2, ...P_T$ be the sequence of time-ordered primitives obtained from $\mathscr{S}$, and $\mathscr{O} = O_1, O_2, ..., O_T$ be the sequence of observations (feature vectors) obtained from the primitives.

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 $G_i = \rho_1, \rho_2, ...\rho_m$ sorted in ascending order.

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., using a single stroke to draw a resistor and the wires on either side of it).

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. 1 shows an example in which two wires (#3 and #6) are interspersed with the transistor.

† Because our domain does not have objects with curves, we only work with line segments. However, our model is general, and supports features computed from any kind of primitive.

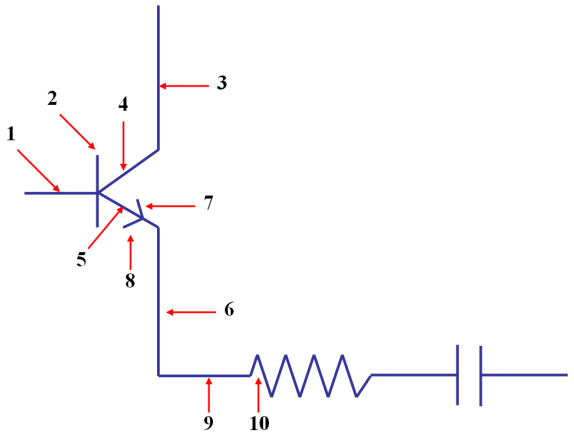

Figure 1: *A diagram illustrating interspersing: The user draws two other objects (wires #3 and #6) over the course of drawing the transistor. Numbers indicate the drawing order.*

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 $G_{\mathscr{B}} = \rho'_1, \rho'_2, ...\rho'_n$. We say that $\mathscr{A}$ is interspersed with $\mathscr{B}$ if $\rho_1 < \rho'_i < \rho_m$ for $1 \leq i \leq n$ and $|G_{\mathscr{A}}| + |G_{\mathscr{B}}| = \rho_m - \rho_1 + 1$. The model that we present handles a more general case of interspersing where $\mathscr{A}$ can be interspersed with multiple objects.

2.2. Desired Features of a Model

The main feature of our model is its ability to handle interspersed drawing behavior. However, we also support the following features identified as important in previous work.

2.2.1. Learning stroke-level and object-level patterns

Stroke orderings used in the course of drawing individual objects naturally contain certain patterns. For example, when drawing arrows, one frequently seen temporal pattern is a long line (the shaft) followed by two shorter lines (parts of the arrow head). These are called **stroke-level patterns** because they capture the probability of seeing a particular sequence of *strokes* with certain properties when sketching an object [SD07].

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 [SD07]. For example, when people draw box-connector diagrams (e.g., organizational charts, linked lists), boxes are drawn before connectors. Our system learns stroke-level and object-level temporal patterns of a domain from examples and uses them in recognition.

2.2.2. Handling multi-stroke objects and variations in encoding length

Users should be able to draw freely. For example, they should be able to draw a square 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). We achieve this by explicitly modeling whether the user has finished drawing an object.

2.2.3. Support for multiple drawing orders

We should be able to accommodate multiple drawing orders instead of just one. For example, it should be possible to draw a square starting with both horizontal and vertical lines. Furthermore, 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.

2.2.4. 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 that 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.

2.2.5. 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 support discrete and real-valued features using appropriate representations such as discrete conditional probabilities and Gaussian mixtures.

3. Recognition System

The requirements listed in section 2.2 collectively impose constraints on the computational model used for sketch recognition. In particular, we want our model to capture the probabilistic relationships between features extracted from a sketch, and to model the stroke-level patterns, object-level patterns as well as the common interspersing behaviors observed in a domain. We achieve this by specifying a joint distribution over a set of random variables that collectively define the dynamics and constraints of our computational model of sketching. Equations describing these constraints are tedious to list and hard to understand at best. Fortunately there is a much simpler way of expressing them using the visual language of *probabilistic graphical models*. Therefore, we describe our models using a class of probabilistic graphical models known as Dynamic Bayesian Networks (DBNs). DBNs have successfully been used to model time series, and are therefore appropriate for problems with a temporal nature. Fig. 2 shows our DBN model for the analog circuits domain. After a brief overview of DBNs, we describe it in detail.

3.1. Dynamic Bayesian Networks and Switching Parents

Because our model of sketch recognition uses Dynamic Bayesian Networks and a relatively unknown feature of DBNs called *switching parents*, we briefly review them both.

3.1.1. 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 extend Bayesian networks and model joint distribution of a set of variables over time by representing the conditional dependencies between them using a pair of Bayesian networks $\langle B_1, B_{\rightarrow} \rangle$. 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.

3.1.2. Switching parents

Our model contains graphical notation (dotted and dashed arrows in Fig. 2) 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 efficiently represent conditional dependencies that change as a function of another node's value [Bil00, GH96, BFGK97]. Fig. 3 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 indicates that:

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

We use switching parents as an efficient mechanism for selecting the process corresponding to the active object in our dynamic model of sketching. Specifically, only one of the nodes **N**, **R**, **C**, **B** and **W** — the stroke-level models for the npn-transistor, resistor, capacitor, battery and the wire objects — is activated at a given time based on the value of the **MUX** node. The rest of our discussion assumes that the reader is comfortable with DBNs and the DBN terminology (an excellent review can be found in [Mur02]).

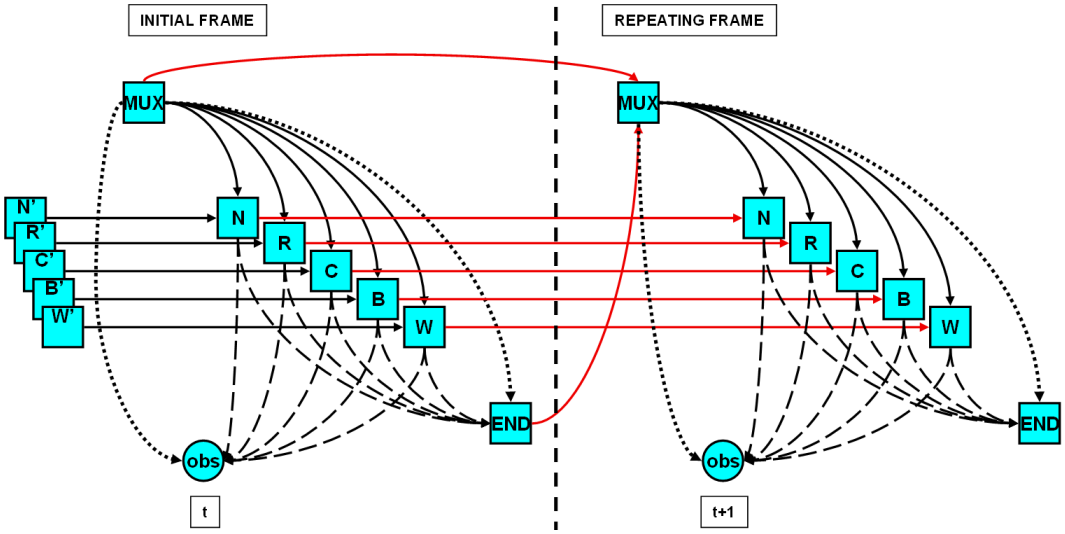

Figure 2: *Dynamic Bayesian network representing our model that handles interspersed drawing.*

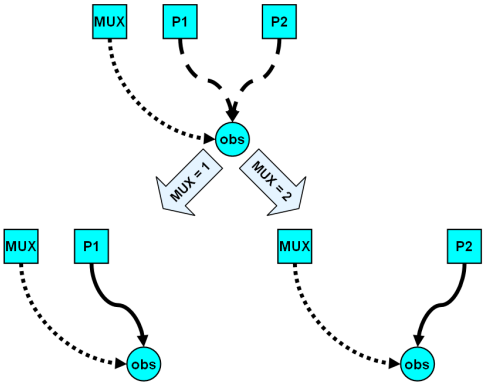

Figure 3: *Illustration of the switching parent mechanism.*

3.2. Sketch Preprocessing and Feature Extraction

As in most sketch recognition systems, we first preprocess an input sketch to extract features that characterize the input. Given an input sketch, we preprocess each stroke and break it into geometric primitives using the early sketch processing toolkit described in [SSD01], then compute a number of geometric features for each primitive.

To facilitate direct comparison of recognition rates with previous work, we use the feature representation suggested by [SD07]. For each primitive P_i, we obtain an observation vector 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} 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 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 $t = 1$).

3.3. Model Description

The Dynamic Bayesian Network specifying our computational model of sketch recognition is shown in Fig. 2. It is important to note that the figure shows only two frames (initial and repeating frames) of the DBN, as this is the conventional way of representing a DBN. As is the case for all DBNs, during classification the network is *unrolled* to produce as many frames as the number of observations.

3.3.1. Description of the nodes

Our network has three groups of nodes. First are the observation nodes $\mathtt{OBS_i}$ that serve as the input to sketch recognition. These are the only observable nodes during recognition. Each observation $\mathtt{OBS_i}$ is a feature vector computed using primitive P_i.

The second group of nodes are the $\mathtt{MUX_i}$ nodes which are multi-valued discrete variables holding our hypothesis of what object P_t is a part of and whether or not the user is interspersing any two objects. Computing the set of assignments to the nodes $\mathtt{MUX_{1:T}}$ that maximize the joint probability of the DBN for a set of observations gives us the classification for each primitive. Hence the $\mathtt{MUX_{1:T}}$ nodes provide the output of sketch recognition.

The third group of nodes are auxiliary nodes. The $\mathtt{END_i}$

node is a discrete boolean node that reflects our belief about whether the user has just completed drawing the current object by drawing the primitive P_i. Explicitly modeling when objects are completed allows us to support multi-stroke objects and variations in the encoding length as mentioned in section 2.2.2. Because the training data is fully labeled, we know when objects are completed. Thus the **END** nodes are observable in training but hidden during recognition. The remaining nodes are always hidden and they capture the statistics of the stroke-level patterns for domain objects. For example, parameters of the **R** node encode the statistics of observations that are typically seen when users draw resistors. Similarly, there are corresponding nodes for each object in our domain (**N**, **C**, **B** and **W** for npn-transistors, capacitors, batteries and wires). The nodes **R′**, **N′**, **C′**, **B′** and **W′** define priors for these nodes, as we explain later. Because we have such nodes for each object class in our domain, we will use the generic notation $\mathbf{C_i}$ to refer to them. So, in the rest of our discussion, we use $\mathbf{C_1}$ = **N**, $\mathbf{C_2}$ = **R**, $\mathbf{C_3}$ = **C**, $\mathbf{C_4}$ = **B** and $\mathbf{C_5}$ = **W**.

3.3.2. Description of the model topology

Recall that the goal of sketch recognition is to assign labels to primitives that constitute valid domain objects. The **MUX** node in our model represents the label of the object being drawn and the information of what objects are interspersed when interspersing occurs. Its cardinality is equal to the sum of the number of object classes and the number of objects that can be interspersed. The semantics of $\mathbf{MUX_t} = i$ is determined by the following:

$$\begin{aligned} &if\ 1 \le i \le |\mathscr{C}| \implies \text{drawing object } i, \mathbf{C_i} \text{ active,} \\ &if\ \quad i > |\mathscr{C}| \implies \text{interspersing } \mathscr{A} \text{ and } \mathscr{B}, \text{ given by} \\ &\qquad\qquad\qquad\quad \text{the function } \mathscr{F}_{int}(i) = \langle \mathscr{A}, \mathscr{B} \rangle. \end{aligned}$$

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 the types of interspersings seen in the training data. For example, in our domain npn-transistors are interspersed with wires, so we define $\mathscr{F}_{int}(6) = \langle \mathbf{N}, \mathbf{W} \rangle$ where $|\mathscr{C}| = 5$.

For each object class that we support, we have a node capturing the dynamics of the stroke-level features (nodes $\mathbf{C}_1$, $\mathbf{C}_2$, ... $\mathbf{C}_{|\mathscr{C}|}$, shown by **N**, **R**, **C**, **B** and **W** in Fig. 2). 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 (i.e., $P_{B_1}(\mathbf{C}_i | Parents(\mathbf{C}_i)) = P_{B_\rightarrow}(\mathbf{C}_i | Parents(\mathbf{C}_i))$. 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:

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

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 whether the user is currently interspersing an object of class $\mathbf{C_j}$ with $\mathbf{C_k}$ given by $\mathscr{F}_{int}(i) = \langle C_j, C_k \rangle$:

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

The interspersing function $\mathscr{F}_{int}(i)$ mapping values of **MUX** to the range $1 \le i \le |\mathscr{C}|$ is constructed prior to the training as mentioned earlier. The 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). The **END** node also has **MUX** 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 $P(\mathbf{C}'_i = 1) = 1$.[‡] 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 simply 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 is updated 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

[‡] 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.

cases correspond to interspersings and $P(\mathtt{MUX_t}=i|\mathtt{MUX_{t-1}}=j,\mathtt{END_{t-1}}=false)>0$ only if we have seen objects of type $\mathtt{C_i}$ being interspersed with another object in the training data.[§] The $\mathtt{C_{i,t}}$ nodes in the repeating frames are conditioned on the $\mathtt{MUX_t}$ and $\mathtt{C_{i,t-1}}$ nodes. The conditional probability table for $P_{B_\rightarrow}(\mathtt{C_{i,t}}|Parents(\mathtt{C_{i,t}}))$ is estimated from the data subject to a few constraints that we specify prior to training in the form of deterministic conditional probability tables [BZ02]. Specifically, we require $P_{B_\rightarrow}(\mathtt{C_{i,t}}=c|\mathtt{MUX_t}=m,\mathtt{C_{i,t-1}}=c')$ to be:

$$
\begin{array}{ll}
1 & \text{if } m \neq i, 1 \leq m \leq |\mathscr{C}|, c = 1 \\
0 & \text{if } m \neq i, 1 \leq m \leq |\mathscr{C}|, c \neq 1) \\
1 & \text{if } m > |\mathscr{C}|, c = c', \text{ and } \mathscr{F}_{int}(m) = \langle C_i, C_* \rangle \\
0 & \text{if } m > |\mathscr{C}|, c \neq c', \text{ and } \mathscr{F}_{int}(m) = \langle C_i, C_* \rangle
\end{array}
$$

where $*$ indicates a wild-card.

These constraints ensure that if the user is drawing an object other than the one associated with the node $\mathtt{C_{i,t}}$, the state of that node 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 $\mathtt{C_i}$ 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.

3.4. Training and Recognition

The $\mathtt{MUX_{1:T}}$, $\mathtt{END_{1:T}}$, and $\mathtt{OBS_{1:T}}$ nodes are observable during training and we estimate the parameters of our DBN using these values. During recognition, only the $\mathtt{OBS_{1:T}}$ values are observable. Using probabilistic inference, we compute the assignments to the $\mathtt{MUX_{1:T}}$ and $\mathtt{END_{1:T}}$ nodes that maximize the joint probability of the network. The $\mathtt{MUX_{1:T}}$ values give us the primitive labels, and $\mathtt{END_{1:T}}$ give us the object boundaries.

As mentioned earlier, we support continuous features. This is done by representing the $\mathtt{OBS_{1:T}}$ nodes using mixtures of Gaussians. We found Gaussians with three components to work well for our domain. We also set the cardinality of the $\mathtt{C_i}$ nodes to be 6 based on our empirical observations and the criteria mentioned in [SD05].

4. Evaluation

We report recognition rates for our model on sketches from the analog circuit diagrams domain to illustrate its performance in absolute terms. We also measure the incremental benefits of modeling interspersing by comparing the correct recognition rates to those obtained using the recognition algorithm reported in [SD07]. That algorithm is an appropriate baseline because it does not handle interspersing, but otherwise supports temporal sketch recognition, albeit using a substantially different architecture.

	Participant ID			
	1	5	6	7
$\bar{x}_b$	89.4	89.8	93.0	84.6
$\bar{x}_i$	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 1: *Mean correct recognition rates for the baseline ($\bar{x}_b$) and our system that models interspersings ($\bar{x}_i$). The percentage reductions in the error rates and maximum error reductions achieved for each user are also listed as percentages (Δ_{err} and $max\Delta$). On average, handling interspersing patterns always improves performance.*

To make the comparison to the baseline meaningful, we ran our system on the same data set, which contains circuit diagrams collected from 8 electrical engineers (10 sketches per participant). Of the eight participants, participants #1, #5, #6 and #7 produced interspersed sketches. Using examples from these subjects, we ran a series of hold-one-out experiments. We trained specialized models for each individual by using sketching examples from that user only.

4.1. Quantitative Results

We trained the baseline model using circuits with no interspersings (because it cannot handle interspersing) and trained our model using all the data. We tested both models using all the examples.

Table 1 shows the average correct recognition rates for each participant obtained using the baseline and our system. 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 always improves performance, and allows 20%-37% of misrecognition errors to be corrected. A paired t-test for the values in Table 1 shows the difference to be statistically significant for $p < 0.05$ and 3 degrees of freedom.

4.2. Qualitative examples and discussion

Fig. 4 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 misrecognitions. 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. 5.

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

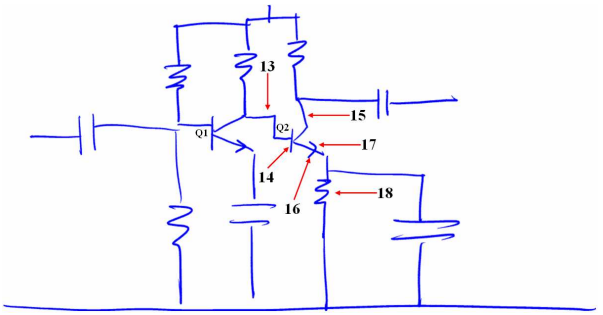

Figure 4: *One of the circuits used in the evaluation. Stroke ordering for a fragment of the circuit is shown by numbers.*

Fig. 5-a shows the interpretation obtained by running the baseline. 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. 5-b shows that both errors are fixed by our model, which 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, it also helps the two wire segments to be classified correctly by using the knowledge that transistors very rarely proceed resistors, $P(\texttt{MUX}_t = \texttt{NPN} \mid \texttt{MUX}_{t-1} = \texttt{RESISTOR}) \approx 0$. This example shows the benefits of modeling both object-level patterns and interspersing.

The incremental benefits of using a more elaborate model may at times appear to be small. Nevertheless, correcting each misclassification requires effort on the part of the user and gets in the way of the main task. Hence, we believe the error reduction rates we have demonstrated are significant in the context of a sketch-based user interface.

5. Related Work

Researchers have reported the existence of interspersed drawing phenomena in other domains, and identified the ability to deal with it as an important challenge [AL07]. However, so far, there has not been much work that specifically addresses the interspersing issue.

As briefly summarized in the introduction, there are a number of systems that use temporal stroke orderings to perform sketch recognition [SD04, ABS04, SD05, SD07]. None of these systems support interspersed drawing. Furthermore, with the exception of [SD07], they do not support stroke sharing across objects and support discrete features only.

There are also approaches to sketch recognition that do not use temporal features (e.g., [AD04, MF02, HD04, KBS04, CS05, GKS04, SVC04]). Some of these algorithms rely on stroke orderings to aid segmentation or to narrow down the search space during matching. These systems either assume no interspersing or suffer high combinatorial search penalties to accommodate interspersings. Others

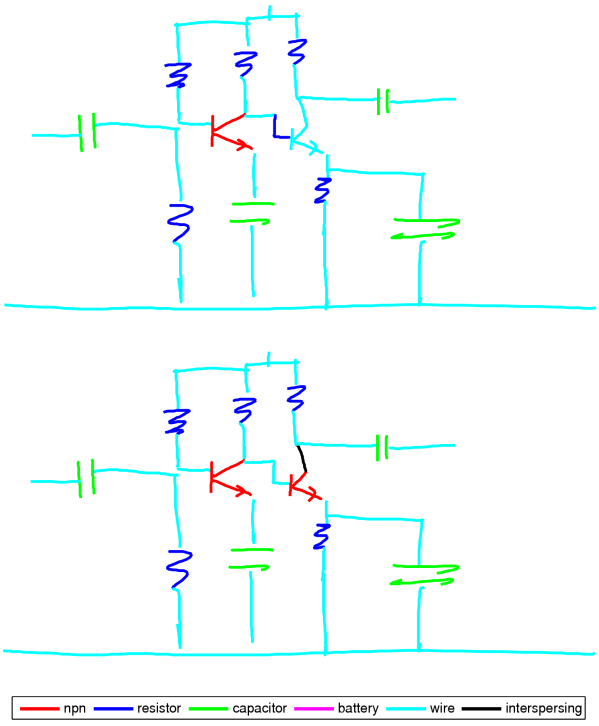

Figure 5: *Interpretations of the interspersed circuit shown in Fig. 4 by the baseline and our system. The baseline encounters a cascade of misclassifications due to interspersing (top) while we correctly identifies the interspersing (bottom).*

use image based representations, hence interspersing is not an issue for these systems. However, to keep their search tractable, they make some assumptions about the objects in the domain (e.g., no objects have no more than 8 strokes, or strokes constituting an object should be located within a certain distance). These assumptions seem to work for flowchart-like drawings where the connectors and objects are sufficiently separated from one another, but the practicality of these algorithms is yet to be demonstrated in general for domains where objects vary in size and shape where assumptions on the object size and scale may not hold.

6. Future Work

We see three immediate lines of further research. First, we need to study drawing behaviors of real users in real world conditions in a variety of domains to get a better understanding of why people intersperse strokes and how the domain affects the interspersing behavior. Our discussion with circuit design experts have revealed that the main cause of interspersings wires with transistors could be a design convention called "following the current" which advocates drawing from top (the positive voltage) towards the bottom of the page (ground). Further research is needed to see how

and why people intersperse strokes in other domains. Researchers have already started exploring this avenue [AL07].

It is also important to explore how properties of the user interface affect the interspersing behavior. For example, our domain contained fewer interspersings compared to the digital logic diagrams domain studied by [AL07]. Part of the difference might be due to reasons intrinsic to the domain, however properties of the sketching user interface might also have an effect (e.g. the existence of an undo button). Affects of the interface design on the degree of temporal regularities and the interspersing behavior is worth exploring.

Finally, we need find ways of handling interspersed drawing within other recognition frameworks. We have adopted a computational model of sketching that defines sketching as a stochastic generative process. This in turn shaped the way we addressed the interspersing problem. Further research is needed to find ways of addressing the problem within other frameworks. It is quite likely that the nature of the solution in each case will depend on the specifics of the recognition algorithm in use (e.g., template matching, image based recognition). Establishing ways of handling interspersing within other recognition frameworks would make it possible to rate the robustness of each system with respect to interspersing and — with a better understanding of which domains show more interspersing — would allow us to choose the appropriate algorithm for a given domain.

References

[ABS04] ANDERSON D., BAILEY C., SKUBIC M.: Hidden markov model symbol recognition for sketch-based interfaces. *AAAI Fall Symposium Series Making Pen-Based Interaction Intelligent and Natural* (2004).

[AD04] ALVARADO C., DAVIS R.: Sketchread: A multi-domain sketch recognition engine. *Proceedings of UIST* (2004).

[AL07] ALVARADO C., LAZZERESCHI M.: Properties of real world digital logic diagrams. *Submitted to 1st International Workshop on Pen-based Learning Technologies* (2007).

[BFGK97] BOUTILIER C., FRIEDMAN N., GOLDSZMIDT M., KOLLER D.: Context-specific independence in bayesian networks. *Uncertainty in Artificial Intelligence* (1997).

[Bil00] BILMES. J. A.: Dynamic bayesian multinets. *In Proc. of the 16th conf. on Uncertainty in Artificial Intelligence.* (2000).

[BZ02] BILMES J., ZWEIG G.: The graphical models toolkit: An open source software system for speech and time-series processing. *IEEE ICASSP. Orlando Florida* (2002).

[CS05] COWANS P. J., SZUMMER M.: A graphical model for simultaneous partitioning and labeling. *AI & Statistics* (January 2005).

[GH96] GEIGER D., HECKERMAN D.: Knowledge representation and inference in similarity networks and bayesian multinets. *Artificial Intelligence, 82:4574* (1996).

[GKS04] GENNARI L., KARA L. B., STAHOVICH T. F.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *AAAI Fall Symposium Series, Making Pen-Based Interaction Intelligent and Natural* (2004).

[HD04] HAMMOND T., DAVIS R.: Automatically transforming shape descriptions for use in sketch recognition. *AAAI* (2004).

[KBS04] KRISHNAPURAM B., BISHOP C., SZUMMER M.: Generative bayesian models for shape recognition. *IWFHR '04, Japan* (2004).

[MF02] MAHONEY J. V., FROMHERZ M. P. J.: Three main concerns in sketch recognition and an approach to addressing them. *AAAI Spring Symposium: Sketch Understanding* (2002).

[Mur02] MURPHY K.: Dynamic bayesian networks. *Chapter in Probabilistic Graphical Models by Michael Jordan* (2002).

[SD04] SIMHON S., DUDEK G.: Sketch interpretation and refinement using statistical models. *Proceedings of the 15th Eurographics Symposium on Rendering (EGSR 04)* (2004).

[SD05] SEZGIN T. M., DAVIS R.: HMM-based efficient sketch recognition. *Proceedings of the 10th international conference on Intelligent User Interfaces, San Diego, California, USA* (2005), 281 – 283.

[SD07] SEZGIN T. M., DAVIS R.: Sketch interpretation using multiscale models of temporal patterns. *IEEE Computer Graphics and Applications* (2007), vol. 27, no. 1, pp. 28–37, Jan/Feb,.

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: Early processing for sketch understanding. *Proceedings of PUI* (2001).

[SVC04] SHILMAN M., VIOLA P., CHELLAPILLA K.: Recognition and grouping of handwritten text in diagrams and equations. *Frontiers in Handwriting Recognition, 2004. IWFHR-9* (2004).

[Tve99] TVERSKY B. G.: What does drawing reveal about thinking? *Invited talk at First International Workshop on Visual and Spatial Reasoning in Design, Cambridge, MA.* (1999).

[vS84] VAN SOMMERS P.: Drawing and cognition. descriptive and experimental studies of graphic production processes. *Cambridge University Press* (1984).