Computers & Graphics 32 (2008) 500–510

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Sketch recognition in interspersed drawings using time-based graphical models

T.M. Sezgin [a,*], R. Davis [b]

[a] College of Engineering, Koç University, Sarıyer, Istanbul 34450, Turkey
[b] Massachusetts Institute of Technology, CSAIL, Cambridge, MA 02139, USA

ARTICLE INFO

Article history:
Received 15 December 2007
Received in revised form
5 April 2008
Accepted 15 May 2008

Keywords:
Temporal sketch recognition
Dynamic Bayesian networks
User interfaces

ABSTRACT

Sketching is a natural mode of interaction used in a variety of settings. With the increasing availability of pen-based computers, sketch recognition has gained attention as an enabling technology for natural pen-based interfaces. Previous work in sketch recognition 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, showing that handling interspersed drawing provides a significant increase in accuracy.

1. Introduction

The activity of sketching is typically (and unconsciously) rather stylized in the sense that people sketch in predictable ways. 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 [1,2].

The existence of ordering patterns during drawing is significant from a recognition perspective, because, as has been demonstrated in a variety of domains, stroke orderings can be used to aid recognition [3–6]. All previous systems, however, make certain assumptions that limit the complexity of the inputs they can accommodate. One system, for example, assumes the scene contains only one object, drawn in a single stroke [4]. Other systems allow recognition in scenes with multiple objects with the restriction that objects or complete object components are drawn using a single stroke [3]. Another approach allows scenes with multiple objects and objects consisting of multiple strokes, but assumes that no objects share strokes [5]. A more recent framework allows stroke sharing under certain conditions and shows how common stroke orderings as well as object orderings can be used for recognition [6].

Even so, one key assumption that all these systems make about free-hand drawing is that people complete each object before moving on to draw the next. Yet real-world data shows this to be untrue. For example, our analysis of free-hand analog electronic circuit diagrams collected from electrical engineers shows it is not uncommon for people to start drawing a new object before completing the current one. This drawing behavior, which we call *interspersed drawing*, occurs in other domains as well [7]. The ability to deal with interspersed drawing is recognized as a major task that sketch recognizers should support [7]. 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.

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

* Corresponding author. +44 7748 727917; fax: +44 1223 334678.
E-mail addresses: mts33@cl.cam.ac.uk, mtsezgin@csail.mit.edu (T.M. Sezgin), davis@csail.mit.edu (R. Davis).

0097-8493/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2008.05.008

2. Problem definition

Informally, the goal of sketch recognition is to segment digital ink drawn by the user, and then classify it by labeling it as one or more of the 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 this domain are wires, resistors, capacitors, npn-transistors and batteries.

2.1. Terminology

We adopt the terminology and notation used in [6]. A *sketch* $\mathscr{S} = S_1, S_2, \ldots, S_N$ 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 objects (e.g., lines, arcs) called *primitives* as part of the preprocessing of the sketch.[1] Let $\mathscr{P} = P_{1:T} = P_1, P_2, \ldots, P_T$ be the sequence of time-ordered primitives obtained from sketch $\mathscr{S}$, and $\mathscr{O} = O_1, O_2, \ldots, O_T$ 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 $\mathscr{C} = \{C_1, C_2, \ldots, C_n\}$, *classification* refers to the task of determining which object each group of primitives represents (e.g., a stick-figure). Segmentation produces K groups $G = G_1, G_2, \ldots, G_K$, and classification gives us the labels for the groups $L = L_1, L_2, \ldots, L_K$, $L_i \in \mathscr{C}$. Each group is defined by the indices of the primitives included in the group $G_i = \rho_1, \rho_2, \ldots, \rho_m$ 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 grouping primitives is more general than a definition based on grouping strokes. By defining segmentation as grouping primitives, we prohibit primitives from being shared across objects, but allow a stroke (which can be composed of multiple primitives) to be part of multiple objects (e.g., using a single stroke to draw a resistor and the wires on either side of it).

We use *interspersing* to mean to the situation where the user starts drawing one object but draws one or more other objects before the first is completed. For example, Fig. 1 shows a circuit fragment in which two wires (#3 and #6) are interspersed with the transistor.

More formally, suppose we have two objects $\mathscr{A}$ and $\mathscr{B}$. Assume the primitive indices for the proper grouping of primitives forming $\mathscr{A}$ and $\mathscr{B}$ are $G_{\mathscr{A}} = \rho_1, \rho_2, \ldots, \rho_m$ and $G_{\mathscr{B}} = \rho'_1, \rho'_2, \ldots, \rho'_n$. We say that $\mathscr{A}$ is interspersed with $\mathscr{B}$ if $\rho_1 < \rho'_i < \rho_m$ for $1 \leqslant i \leqslant n$ and $|G_{\mathscr{A}}| + |G_{\mathscr{B}}| = \rho_m - \rho_1 + 1$. The model we present is in fact able to handle a more general case of interspersing where $\mathscr{A}$ is 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 five 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 [6].

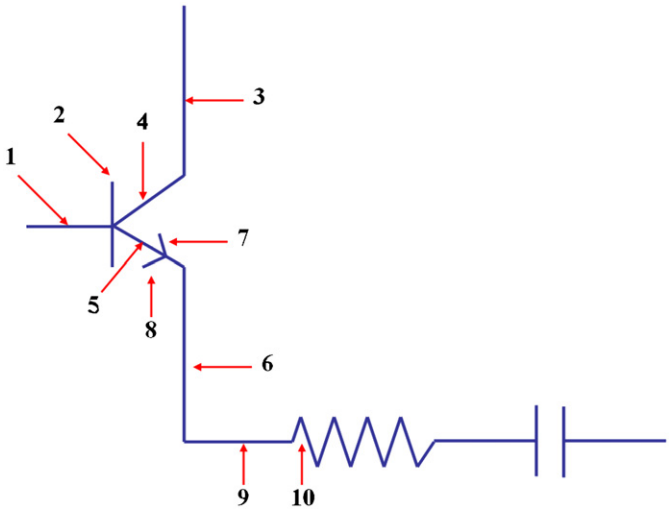

Fig. 1. A diagram illustrating interspersing: The user draws two other objects (wires made from primitives #3 and #6) over the course of drawing the transistor (primitives #2, #4, #5, #7, #8). Numbers indicate the primitive drawing order.

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 [6]. For example, when people draw box-connector diagrams (e.g., organizational charts, linked lists), boxes are typically drawn before connectors.

Our system learns both 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 one, two, three or four strokes, or draw a resistor with different numbers of humps (thus generating encodings of the input with different numbers of observations). 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 either horizontal or vertical lines. Furthermore, the system ought to learn about the user, learning for example whether the user uses one drawing order more frequently than others. This requires training and classification methods that can use such information. We achieve this by adopting a probabilistic machine learning framework where parameters are estimated to capture the statistics of the training data.

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

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

are preferred. Our framework based on DBNs satisfies this criterion.

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. In response, our algorithms support both 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 and 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 DBNs. DBNs have successfully been used to model time series, and are therefore appropriate for problems with a temporal nature. We start with a brief overview of DBNs, then describe our model in detail.

3.1. DBNs and switching parents

Because our model of sketch recognition uses DBNs 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 $Z = \{Z_1, \ldots, Z_n\}$ 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 variables 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 both 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. 3) 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 [8–10].

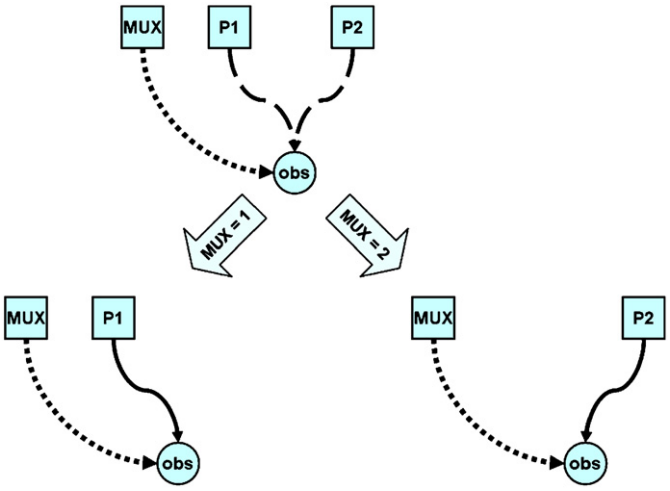

Fig. 2. Illustration of the switching parent mechanism.

Fig. 2 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

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

We use switching parents as an efficient mechanism for selecting the process corresponding to the active object (i.e., the object currently being drawn) in our dynamic model of sketching. Specifically, in our model, shown in Fig. 3, only one of the object models (an npn-transistor, resistor, capacitor, battery or wire, indicated by **N**, **R**, **C**, **B** and **W**) is activated at any 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 of DBNs can be found in [11].

3.2. Model description

The DBN specifying our computational model of sketch recognition is shown in Fig. 3. The figure shows only two frames of the DBN (the initial and repeating frames), 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. Fig. 4 shows a small circuit fragment and an example of the unrolled DBN.

3.2.1. Description of the nodes

Our network has three groups of nodes. First are the observation nodes $\mathbf{OBS_i}$ that serve as the input to sketch recognition. These are the only observable nodes during recognition. Each observation $\mathbf{OBS_i}$ is a feature vector computed based on the ith primitive P_i extracted from the input sketch via feature extraction.

The second group of nodes are the $\mathbf{MUX_i}$ 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. The classification for each primitive is obtained by computing the set of assignments to the nodes $\mathbf{MUX_{1:T}}$ that maximize the joint probability of the DBN for a set of observations. Hence the $\mathbf{MUX_{1:T}}$ nodes provide the output of sketch recognition.

The third group of nodes are auxiliary nodes. The $\mathbf{END_i}$ node is a discrete boolean node that reflects our belief about whether the

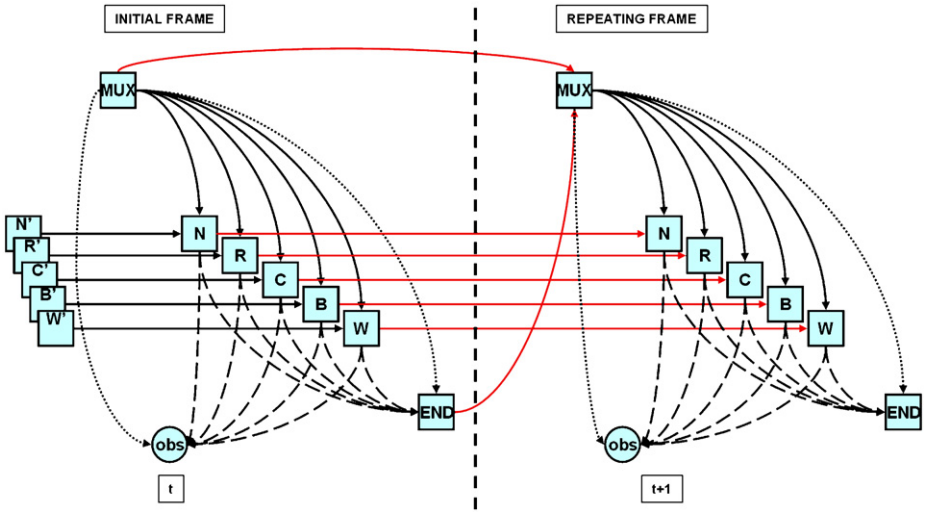

Fig. 3. Dynamic Bayesian network representing our model that handles interspersed drawing.

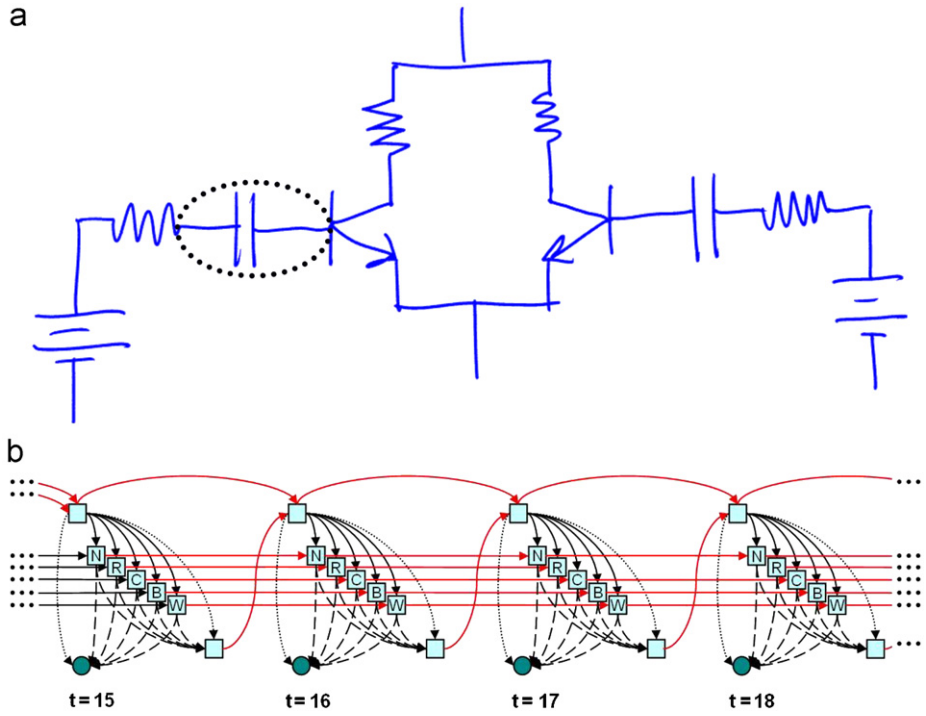

Fig. 4. A circuit diagram and a fragment indicated by the dotted circle (a). The unrolled dynamic Bayesian network fragment corresponding to the circuit fragment (b). During recognition, only the **OBS** nodes are observable—indicated by darker color.

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 mentioned in Section 2.2.2. Because the training data are fully labeled, during training we know when objects are completed. Thus the $\mathbf{END_i}$ 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 below. For ease of reference, we will use the generic notation $\mathbf{C_i}$ to refer to each of these object classes in our domain ($\mathbf{C_1} = \mathbf{N}$, $\mathbf{C_2} = \mathbf{R}$, $\mathbf{C_3} = \mathbf{C}$, $\mathbf{C_4} = \mathbf{B}$ and $\mathbf{C_5} = \mathbf{W}$).

3.2.2. 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:

if $1 \leqslant i \leqslant |\mathscr{C}| \Longrightarrow$ drawing object i, $\mathbf{C_i}$ active,
if $i > |\mathscr{C}| \Longrightarrow$ interspersing $\mathscr{A}$ and $\mathscr{B}$, given by
the function $\mathscr{F}_{int}(i) = \langle \mathscr{A}, \mathscr{B} \rangle$

In our current domain, there are five object classes, i.e., $|\mathscr{C}| = 5$. 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 were interspersed with wires, so we define $\mathscr{F}_{int}(6) = \langle \mathbf{N}, \mathbf{W} \rangle$. Construction of the interspersing function $\mathscr{F}_{int}$ can be done automatically by a simple analysis of the training data that detects interspersings.

For each object class there is a node capturing the dynamics of the stroke-level features for that object (nodes $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{|\mathscr{C}|}$, shown by **N**, **R**, **C**, **B** and **W** in Fig. 3). The "primed" nodes $(\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}'_{|\mathscr{C}|})$ 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., $|\mathbf{C}'_i| = |\mathbf{C}_i|$).

3.2.2.1. Conditional dependencies for the initial frame. The **MUX** node in the initial frame has no parents; 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:

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

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 $\mathscr{F}_{int}(i) = \langle C_j, C_k \rangle$:

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

The interspersing function $\mathscr{F}_{int}(i)$ mapping values of **MUX** to the range $1 \leqslant i \leqslant |\mathscr{C}|$ is constructed prior to training as explained above. The use of switching parents in this fashion also allows us to learn and share a single model for objects that are interspersed: 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: $P(\mathbf{END} | \mathbf{MUX} = i, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n) = P(\mathbf{END} | \mathbf{MUX} = i, \mathbf{C}_i)$.

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 $P(\mathbf{C}'_i = 1) = 1$.[2] 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 $\mathbf{C}_{i,t-1}$ for the initial update. 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 gives us an efficient way of sharing probability distributions through a parameter tying mechanism [12]. $\mathbf{C_i}$ nodes that are not selected by the **MUX** node in the initial frame simply copy values from $\mathbf{C}'_i$.

3.2.2.2. 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 cases correspond to interspersings and $P(\mathbf{MUX_t} = i | \mathbf{MUX_{t-1}} = j, \mathbf{END_{t-1}} = false) > 0$ only if we have seen objects of type $\mathbf{C_j}$ being interspersed with other objects in the training data and $i > |\mathscr{C}|$.[3] The $\mathbf{C_{i,t}}$ nodes in the repeating frames are conditioned on the $\mathbf{MUX_t}$ and $\mathbf{C_{i,t-1}}$ nodes. The conditional probability table 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 conditional probability tables [12]. Specifically, we require that:

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

where $*$ indicates a wild card.

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

3.3. Sketch preprocessing and feature extraction

As in most sketch recognition systems, we preprocess an input sketch to extract features that characterize the input. Given an input sketch, we break each stroke into geometric primitives using the early sketch processing toolkit described in [13], 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 [6]. For each primitive P_i, we obtain an observation vector O_t represented as a five-tuple $(\boldsymbol{l}_t, \Delta l_t, \boldsymbol{\theta}_t, \Delta\boldsymbol{\theta}_t, \mathrm{sgn}_t)$ where:

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

[2] 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.

[3] For $1 \leqslant i, j \leqslant |\mathscr{C}|$ 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 table.

The value of $\Delta\theta_t$ is given by the magnitude of the cross product $\vec{u} \times \vec{v}$ of vectors $\vec{u}$, $\vec{v}$, 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 $\vec{u} \times \vec{v}$ and 1 for positive values or $t = 1$.

3.4. Training and recognition

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

As mentioned earlier, we support continuous features. This is done by representing the $\mathbf{OBS_{1:T}}$ nodes using mixtures of Gaussians [12]. We found Gaussians with three components to work well for our domain. We also set the cardinality of the $\mathbf{C_i}$ nodes to be 6 based on the criteria mentioned in [5].

3.5. Implementation details

We used the Graphical Models Toolkit (GMTK [12]) to implement our model. We chose GMTK, because it supports a number of efficient representation mechanisms that we take advantage of such as parameter tying and deterministic conditional probability tables. As a practical matter, we were limited to work with the Frontier Algorithm—an exact inference algorithm—as it is the only one supported by GMTK [36].[4] GMTK allows one to specify graphical models using the GMTK model specification language where the model structure, types of the probabilistic dependencies, and the set of observable and hidden model variables is specified prior to model training.

In GMTK, the training and inference algorithms operate on a triangulated version of the model graph, and as runtime complexity depends on the goodness of the triangulation, obtaining a good triangulated graph is important. Unfortunately, computing the optimal triangulation is known to be NP-hard [14]; we instead resort to GMTKs triangulation heuristics to search for good triangulations. Although this search takes a long time, (e.g., we allow it to run for on the order of 6 h on a 2.6 GHz machine), once a good triangulation is found, it can be reused for recognizing different sketch instances as long as the model remains the same (i.e., the model topology, node cardinalities, and the set of observed/hidden nodes remain the same).

3.6. Design considerations

There are a number of design choices that we made while building our model that might need reconsideration if it were applied to a different domain.

3.6.1. Independence assumptions

One design question that needs consideration when building this model is whether or not the state transition probabilities for the $\mathbf{MUX_{t+1}}$ node depend on the value of $\mathbf{C_i}$ from time t. In the formulation above, we assumed that the point at which the user starts interspersing is conditionally independent of the state of the current object process (e.g., the probability of interspersing a wire is independent of how much of the transistor is drawn). This may not be the case for all domains. In domains where beginning an interspersing is more likely for certain partial drawings, it would be appropriate to add a conditional dependency arc from $\mathbf{C_i}$ at time t to $\mathbf{MUX_{t+1}}$ that switches on when the interspersing begins. Obviously when such a conditional dependence is introduced, more examples will be necessary to estimate model parameters without overfitting. If not enough training examples are supplied, interspersings that begin at states not covered in the training data will receive zero probabilities. This is a sparse data problem and can be handled using regularization techniques. A discussion and an example of how this can be done using pseudo-counts and Dirichlet priors can be found in [15,16].

3.6.2. Multiple interspersings

A second design question that needs consideration is whether we allow more than two objects to be partially drawn at any given time (multiple-interspersings) or whether we allow interspersings between objects of the same type (same-class interspersings). We framed our approach for handling interspersing based on our observations about domains that we studied in current and previous work (e.g., finite state machines, UML diagrams, Course of Action Diagrams, stick-figures and emotions [5,6]), where we did not observe multiple or same-class interspersings. As a result our model does not handle such cases. It would be an interesting exercise to investigate the drawing patterns in other domains to see if any of them could benefit from having more general models of interspersing. Work in [17] suggests two alternative models for such domains.

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 [6]. That algorithm is an appropriate baseline because it does not handle interspersing, but otherwise supports temporal sketch recognition, albeit using a substantially different architecture. 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.

4.1. Quantitative results

We trained the baseline model using circuits with no interspersings (because the baseline cannot handle interspersing) and trained our model using all the data. We tested both models using all the examples. To facilitate comparison with previous work, we trained user-specific (i.e., personalized) models.

Table 1 shows the average correct recognition rates for each participant obtained using the baseline and our system, computed by leave-one-out cross validation. The table also shows the average and maximum percentage reductions in the error rates 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] There are a number of alternative exact inference algorithms, and even a number of approximate inference algorithms with better algorithmic time complexities. Unfortunately, these are not supported by GMTK.

Table 1
Mean correct recognition rates for the baseline ($\bar{x}_b$) and our system that models interspersings ($\bar{x}_i$)

	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
maxΔ	61.5	33.3	100.0	54.5

The percentage reductions in the error rates and maximum error reductions achieved for each user (Δ_{err} and maxΔ). On average, handling interspersing patterns always improves performance.

4.2. Quantifying errors per-interspersing

The percentages presented above do not show the full extent of the consequences 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 seen if we look at the number of misclassified primitives per interspersing.

To measure this we ran both the baseline and our model on a control group that contained 10 sketches with no interspersings, 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 the baseline with the non-interspersed data and supplied our model with the additional interspersed examples. Table 2 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 misclassified by the baseline and by our model. The last two rows show the total number 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, introducing interspersings causes as many as 16 primitives to be misclassified. Considering that a transistor has five primitives, this 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.

4.3. Effects of increased model complexity

Our model is more powerful and also more complex than the baseline, because it can model interspersings. The previous subsection quantifies the performance improvement gained by adopting a more complex model, but a related question is whether the added complexity hurts recognition rates if the test data contains no interspersings. To answer this question, we trained our baseline system (which does not model interspersings) using non-interspersed data and supplied our model with additional interspersed examples.[5]

We ran both the baseline and our model on non-interspersed sketches. As before, all the training and testing was done using leave-one-out cross validation. We used the Wilcoxon matched-pairs signed-ranks test to compare the performance of the two models with the null hypothesis that the paired observations come from the same distributions. The Wilcoxon test fails to reject the null hypothesis ($W+ = 17,\ W- = 4,\ N = 6,\ p \leqslant 0.22$). We regard this result as promising evidence that the complexity of our method does not hurt recognition for non-interspersed drawings.

4.4. Analysis of training time

Parameter estimation is done using the expectation–maximization (EM) algorithm. For a given training set, parameters usually converge to their final values within 20–30 EM iterations. As it is usually the case, we limit the maximum number of allowed EM iterations and automatically stop parameter estimation if an upper bound of 100 EM iterations is reached.

We measured the time taken by a single EM iteration as a function of the training data size by increasing the number of sketches in the training data set from 1 to 20, each containing 80–109 primitives. Fig. 5 shows the time spent for a single EM iteration in seconds as a function of the number of primitives. As seen in the graph, over the range shown the training time scales linearly with respect to the problem size.

4.5. Analysis of the runtime

The runtime of recognition algorithms is one of the main concerns in sketch recognition [18]. In fact, efficiency concerns has been one of the main motivations for exploring temporal recognition algorithms [5,6]. In response, we measured runtime performance at two different levels of granularity: primitive-level and sketch-level.

Primitive-level runtime performance measures the marginal cost of increasing the input size by one primitive. Fig. 6 shows the time spent for inference as a function of the number primitives, on a Linux machine with a 1.6 GHZ Intel processor and 4 gigabytes of memory. As seen in the graph, over the range shown the runtime scales linearly with respect to the problem size. This is consistent with the algorithmic complexity of inference in our DBN, which is $O(T)$ where T is the length of the observation sequence. The marginal cost of processing a single primitive, given by the slope of the runtime line (colored blue), is 0.185 s/primitive. This easily allows 5–6 primitives to be processed per second, which is roughly equivalent to processing 2–3 capacitors, a typical resistor or a transistor per second.

Sketch-level runtime performance measures the amount of time needed to interpret a complete sketch and varies as a function of the number primitives obtained from the sketch. Table 3 shows sketch-level runtime performance of our system. The first row shows the number of total primitives. The second and third rows show the amount of time spent by our system for interpreting each sketch and time that was taken to draw each sketch. As seen in the table, time required for interpretation is only a small fraction of the drawing time.

Finally, as mentioned earlier, in addition to the sketch size, the computational complexity of inference depends on many other factors such as the goodness of the triangulation, the inference

[5] Note that if our model is trained with only non-interspersed examples, then it behaves precisely like the baseline.

Table 2
This table shows the number of misrecognized primitives per interspersing

	Sketch ID										$\bar{x}$
	1	2	3	4	5	6	7	8	9	10	
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

The first two rows show the number of total primitives in each sketch and the number of added interspersings. Next two rows show the number of primitives misclassified 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.

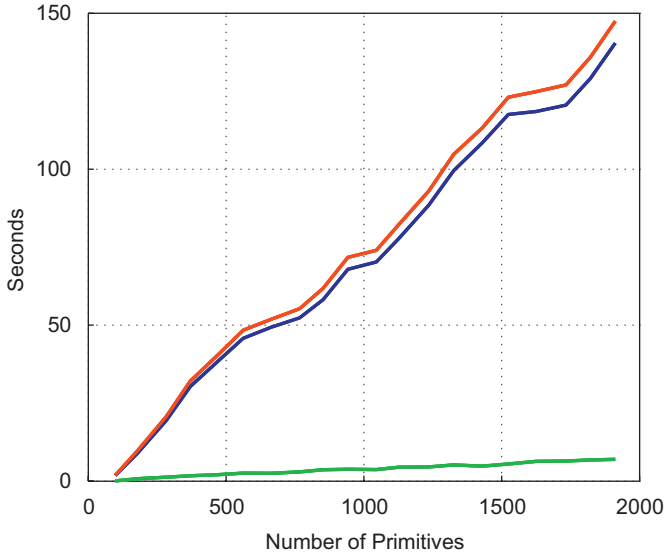

Fig. 5. Time spent for a single EM iteration in seconds as a function of the number of primitives in the training data. The blue line represents the time spent for a single EM iteration. The green line represents system overhead. Red line is the sum of the two.

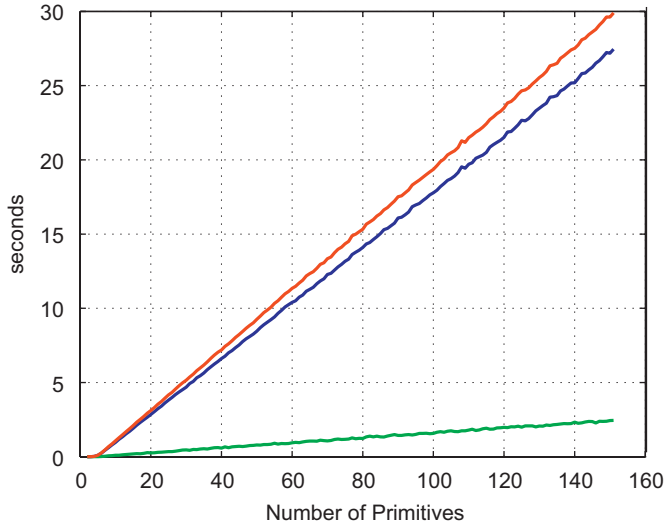

Fig. 6. Runtime plotted as a function of the number of primitives extracted from a sketch. The blue line represents the time spent for inference. The green line represents system overhead. Red line is the sum of the two.

algorithm used, and the state space of the process being modeled (as determined by $|\mathscr{C}|$). The large number of exact and approximate inference algorithms and the NP-hard nature of finding optimal triangulations makes an exhaustive exploration of all factors affecting the runtime impractical. Nevertheless, we have successfully run our model for $|\mathscr{C}| = 10$. Table 4 lists the average processing time per primitive obtained over 10 iterations for $T = 90$. As seen in the table, the mean time needed to process a single primitive is still under a second.[6]

4.6. Qualitative examples and discussion

Fig. 7 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. 8. Fig. 8a 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. 8b shows that both errors are fixed by our model, which learned that with probability 0.14 wires can be interspersed while drawing transistors. This aspect of the model not only allows the transistor to be identified correctly, it also helps the two wire segments to be classified correctly, using the knowledge that transistors very rarely follow resistors, $P(\mathbf{MUX_t} = \mathbf{NPN}|\mathbf{MUX_{t-1}} = \mathbf{RESISTOR}) \approx 0$. This example shows the benefits of modeling both object-level patterns and interspersing.

The incremental benefits of using a more elaborate model may 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

[6] Note that, the variation in the runtime is also affected by the change in the goodness of the triangulation, which we necessarily have to recompute after modifying the model.

Table 3
Sketch-level runtime performance

	Sketch ID										$\bar{x}$
	1	2	3	4	5	6	7	8	9	10	
Total primitives	90	85	104	101	92	100	87	105	80	98	94.2
Interpretation time	16.1	15.3	18.7	18.1	16.4	17.9	15.5	18.9	14.3	17.6	16.9
Drawing time	112.1	67.4	85.8	84.7	84.4	84.1	57.1	62.0	46.8	88.1	77.3
IT/DT ratio	0.14	0.22	0.21	0.21	0.19	0.21	0.27	0.30	0.30	0.20	0.22

The first row shows the number of total primitives. The remaining rows respectively list the time spent by our system for interpreting each sketch, the time taken to draw them, and the ratio of the two.

Table 4
Mean time spent per primitive in milliseconds and the corresponding standard deviations for 10 runs with $|\mathscr{C}| = 10$

	Runtime	System overhead	Total
Mean	57.02	4.48	61.50
Std. deviation	0.67	0.32	0.62

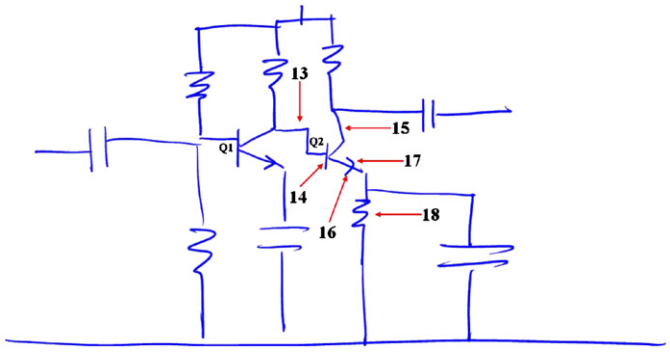

Fig. 7. One of the circuits used in the evaluation. Stroke ordering for a fragment of the circuit is shown by numbers.

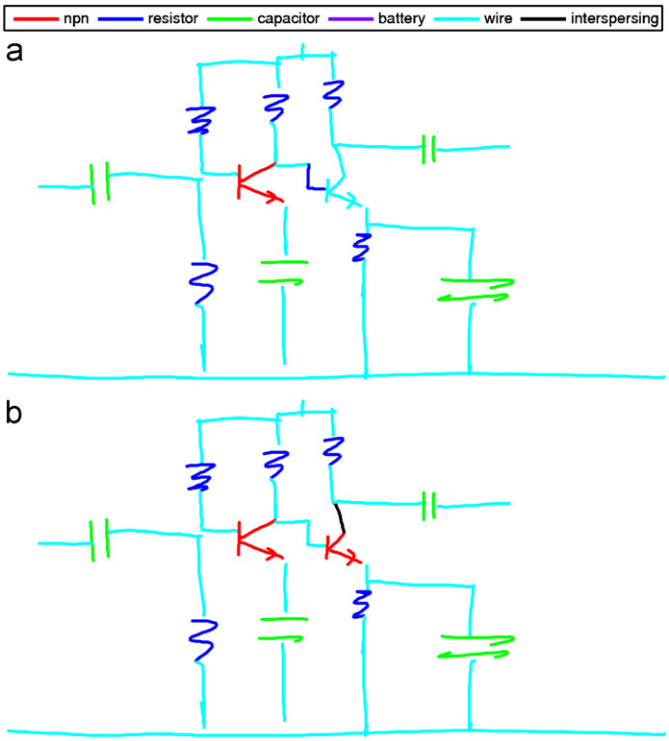

Fig. 8. Interpretations of the interspersed circuit shown in Fig. 7 by the baseline and our system. The baseline encounters a cascade of misclassifications due to interspersing (top) while we correctly identify the interspersing (bottom).

to deal with it as an important challenge for building robust free-sketch recognition systems [7]. However, so far, there has not been much work specifically addressing the interspersing issue. This section reviews various recognition methods (symbol recognizers, spatial and temporal sketch recognizers) and provides a discussion of their strengths and weaknesses compared to our method.

5.1. Symbol recognition systems

The work on symbol recognition includes gesture recognition systems. Symbol recognizers perform isolated object recognition based on geometric and image-based descriptors [19–24]. These systems assume that only a single instance of a properly segmented object is given as input, hence do not address the sketch segmentation and interspersing issues. There are also systems that assume the scene contains an object of a known class, and define sketch recognition as the identification of its subcomponents [25]. This is unlike the definition of sketch recognition that we adopt here.

5.2. Spatial recognition systems

The vast majority of sketch recognition systems do not use temporal features, hence we refer to them as spatial recognition systems. Our approach to recognition complements these by its ability to exploit temporal information.

Template-based approaches to recognition are the most prevalent spatial recognition method [18,26,27]. These methods require object descriptions (also referred to as templates or object models) that describe the domain and the objects. One limitation of template-based methods is scalability of runtime performance. Scalability issues arise from the combinatorial cost of template matching and are typically addressed by pruning the search space of matches, at the cost of overlooking valid interpretations.[7] Another limitation of these systems is the tedious and labor-intensive nature of producing object descriptions. Generating object descriptions generally requires an expert to study the domain objects, and iteratively refine a set of constraints that work well for real data. Recent work on learning object descriptions from examples might remedy this shortcoming, if its applicability to realistic datasets can be demonstrated [28,29]. By contrast, our method learns object models automatically from

[7] Note that search space pruning techniques can also be incorporated into our framework by setting the probability of interpretations with low probabilities to zero, or by committing to interpretations with high probabilities. Although this is not directly supported by GMTK, [17] describes how such information can be included using by making hidden nodes observed during inference.

examples and scales linearly with the sketch size. It is also worth noting that two of the systems mentioned above, [26,27], were designed and evaluated for online recognition scenarios in the sense that their interpretations were updated after each stroke is added. Thus, their performance implicitly depends on the drawing order, and—perhaps to a greater extent—on the existence (or lack thereof) of interspersings. We believe it would be interesting to investigate the effect of interspersing on the matching time and recognition rates of these methods. This would give us a better understanding of how these methods deal with real data.

Szummer et al. [30] describe a generative Bayesian framework for shape recognition. They use dynamic programming to achieve segmentation and classification simultaneously. The major strength of their approach is the ability to recognize messy drawings using single examples by assuming a Gaussian noise model. As was the case for some of the approaches described above, the authors make certain assumptions to keep the search for the optimal segmentation tractable. Specifically they assume that subsets of stroke fragments considered during segmentation contain no more than seven straight line segments. This might be a severe limitation in domains with moderately complex objects, such as the sloppy stick-figures considered by Mahoney et al. [18,31] or the domain of military course of action diagrams.

Later work by Szummer et al. [32] describes a graphical model for simultaneously segmenting and labeling organizational charts. In that work, they demonstrate that labeling and segmenting a sketch simultaneously improves recognition performance. They use a set of 98 features, some of which they claim to be temporal, but there is no information on the specifics of any of the features. They also make a number of simplifying assumptions to keep the search tractable (e.g., maximum clique size for recognition, hard thresholds for distance constraints).

In [33], Kara et al. present a circuit diagram recognition system that does segmentation using a heuristic called "ink-density" and runs isolated symbol recognizers to generate an interpretation. Their system has a number of strengths such as fast recognition, support for multi-stroke objects and multi-object strokes and arbitrary stroke orderings within each object. On the other hand the segmentation algorithm used in this work does not handle interspersed drawing, so the user is required to finish an object before starting a new one.

Shilman et al. present a discriminative recognition framework combining stroke-based and image-based features [34]. Their features are purely spatial and image-based, hence interspersing is not an issue. However, as with some systems above, to keep their search tractable they assume that objects have no more than eight strokes, and that strokes constituting an object are located within a prespecified distance.

5.3. Temporal recognition systems

As noted briefly in the Introduction, a number of systems use temporal stroke orderings to perform sketch recognition [3–6], but with the exception of [6], none of these systems support stroke sharing and continuous feature representations as the system reported here does. The work in [6], however, does not support interspersing, motivating the work presented here.

Work by Anderson et al. describes a symbol recognition method that models the temporal regularities using hidden Markov models and chain-code-based features [4]. They assume isolated objects, hence do not perform segmentation.

Simhon and Dudek present a sketch interpretation and curve refinement system using a hierarchical hidden Markov model (HHMM) [3]. 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. By contrast, 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. Also, they do not support interspersed drawing, multi-stroke objects and real-valued continuous features.

The HMM-based sketch recognition method presented in [5] improves upon the work by Simhon and Dudek by allowing multi-stroke objects through combining matching results from multiple HMMs within a dynamic programming framework. However this work does not model object-level patterns, does not support continuous features, and assumes no interspersing.

6. Future work

As with other temporal recognition systems, our temporal model 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, not shape. Although it is possible to augment our feature sets to include limited shape information extracted from bigrams or trigrams of primitives, integrating a comprehensive list of measurements reflecting constraints between arbitrary groups of strokes (as in [26]) is currently not possible if we would like to accommodate interspersed drawing.[8] Therefore, combining spatial and temporal features in the presence of interspersing is an interesting avenue to pursue.

Studying drawing behaviors of real users in real world conditions for a variety of domains would give us a better understanding of why people intersperse strokes and how the domain affects the interspersing behavior. This might help us design better recognition systems. It would also be relevant for psychologists who are interested in the cognitive modeling of the drawing process. Our discussions 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, as has been initially explored in [7].

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 [7]. Part of the difference might be due to reasons intrinsic to the domain, but properties of the user interface might also have an effect. For example, systems that use explicit buttons or timeouts for indicating segmentation (e.g., [23,35]) might instill and reinforce non-interspersed drawing habits that may carry over to applications which do not require such behavior. Effects of the interface design on the degree of temporal regularities and the interspersing behavior is worth exploring.

Finally, we need to 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 for each framework will depend on the specifics of the recognition algorithm in use (e.g., template matching, image based recognition). Establishing

[8] If we assume no interspersings, an approach based on dynamic programming can be used to combine spatial and temporal constraints as outlined in [17].

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.

Acknowledgments

Support for this work was provided by the MIT Oxygen Project and the MIT/Microsoft iCampus program.

References

[1] Tversky BG. What does drawing reveal about thinking? Invited talk at first international workshop on visual and spatial reasoning in design, Cambridge, MA; 1999.
[2] van Sommers P. Drawing and cognition descriptive and experimental studies of graphic production processes. Cambridge University Press; 1984.
[3] Simhon S, Dudek G. "Sketch Interpretation and Refinement Using Statistical Models," Proc. 15th Eurographics Symp. Rendering (EGSR 04), Eurographics Assoc., 2004, pp. 23–32.
[4] Anderson D, Bailey C, Skubic M. Hidden markov model symbol recognition for sketch-based interfaces. In: AAAI fall symposium series making pen-based interaction intelligent and natural; 2004.
[5] Sezgin TM, Davis R. HMM-based efficient sketch recognition. In: Proceedings of the 10th international conference on intelligent user interfaces, San Diego, CA, USA; 2005, p. 281–3.
[6] Sezgin TM, Davis R. Sketch interpretation using multiscale models of temporal patterns. IEEE Computer Graphics and Applications 2007;27(1): 28–37.
[7] Alvarado C, Lazzereschi M. Properties of real world digital logic diagrams. Submitted to first international workshop on pen-based learning technologies; 2007.
[8] Bilmes JA. Dynamic Bayesian multinets. In: Proceedings of the 16th conference on uncertainty in artificial intelligence; 2000.
[9] Geiger D, Heckerman D. Knowledge representation and inference in similarity networks and Bayesian multinets. Artificial Intelligence 1996;82:4574.
[10] Boutilier C, Friedman N, Goldszmidt M, Koller D. Context-specific independence in Bayesian networks. In Proc. 12'th Conf. Uncertainty in Artificial Intelligence, pp. 115–123. 1996.
[11] Murphy K. Dynamic Bayesian networks. In: Jordan M, editor. Probabilistic graphical models; 2002.
[12] Bilmes J, Zweig G. The graphical models toolkit: an open source software system for speech and time-series processing. In: IEEE ICASSP, Orlando, FL; 2002.
[13] Sezgin TM, Stahovich T, Davis R. Sketch based interfaces: early processing for sketch understanding. In: Proceedings of PUI; 2001.
[14] Murphy K. Dynamic Bayesian networks: representation, inference and learning. PhD thesis, UC Berkeley; 2002.
[15] Murphy K, Mian S. Modelling gene expression data using dynamic Bayesian networks. Technical Report, Computer Science Division, University of California, Berkeley, CA; 1999.
[16] Koller D, Fratkina R. Using learning for approximation in stochastic processes. In: Proceedings of the 15th international conference on machine learning; 1998.
[17] Sezgin TM. Online sketch recognition from a dynamic perspective. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology; June 2006.
[18] Mahoney JV, Fromherz MPJ. Three main concerns in sketch recognition and an approach to addressing them. In: AAAI spring symposium: sketch understanding; 2002.
[19] Rubine D. Specifying gestures by example. Computer Graphics 1991; 25(4):329–37.
[20] Apte A, Vo V, Kimura TD. Recognizing multistroke geometric shapes: an experimental evaluation. In: Proceedings of the sixth annual ACM symposium on user interface software and technology, Atlanta, Georgia, United States; 1993, p. 121–8.
[21] Sun Z, Liu W, Peng B, Zhang B, Jianyong S. User adaptation for online sketchy shape recognition. In: GREC 2003, Barcelona, Spain, July 30–31; 2003 (Revised Selected Papers. Graphics recognition: recent advances and perspectives, In: Fifth international workshop. Lecture notes in computer science, vol. 3088, Springer, Heidelberg: Berlin; 2004. p. 305. ISSN: 0302-9743.)
[22] Watanabe T, Sugawara K, Sugihara H. A new pattern representation scheme using data compression. IEEE Transactions on Pattern Analysis and Machine Intelligence 2002;24(5):579–90.
[23] Hse H, Newton AR. Recognition and beautification of multi-stroke symbols. In: AAAI fall symposium series 2004; 2004.
[24] Kara LB, Stahovich TF. An image-based trainable symbol recognizer for sketch-based interfaces. In: AAAI fall symposium series 2004; 2004.
[25] Sharon D, van de Panne M. Constellation models for sketch recognition. In: Eurographics workshop on sketch based interfaces and modeling; 2006.
[26] Alvarado C, Davis R. Sketchread: a multi-domain sketch recognition engine. In: Proceedings of UIST; 2004.
[27] Shilman M, Pasula H, Russel S, Newton R. Statistical visual language models for ink parsing. In: AAAI spring symposium: sketch understanding, Stanford CA, March 25–27, 2002.
[28] Hammond T, Davis R. Interactive learning of structural shape descriptions from automatically generated near-miss examples. In: Intelligent user interfaces (IUI); 2006.
[29] Veselova O. Perceptually based learning of shape descriptions. Masters Thesis, Cambridge: MIT; 2003.
[30] Krishnapuram B. Bishop C. Szummer M. Generative models and Bayesian model comparison for shape recognition. In: IWFHR '04, Japan; 2004.
[31] Mahoney JV, Fromherz MPJ. Handling ambiguity in constraint-based recognition of stick figure sketches. In: SPIE document recognition and retrieval IX conference, San Jose, CA; January 2002.
[32] Cowans PJ, Szummer M. A graphical model for simultaneous partitioning and labeling. In: AI & Statistics; January 2005.
[33] Gennari L, Kara LB, Stahovich TF. Combining geometry and domain knowledge to interpret hand-drawn diagrams. In: AAAI fall symposium series, making pen-based interaction intelligent and natural; 2004.
[34] Shilman M, Viola P. Spatial Recognition and Grouping of Text and Graphics. Eurographics Workshop on Sketch-Based Interfaces and Modeling (Eurographics SBIM) 2004.
[35] Cohen P, Johnston M, McGee D, Oviatt S, Pittman J, Smith I, et al. Quickset: multimodal interaction for distributed applications. In: ACM MM 1997; 1997.
[36] Zweig G, Russell S. Probabilistic Modeling with Bayesian Networks for Automatic Speech Recognition. Australian Journal of Intelligent Information Processing Systems 1999;5(4):253–60.