HMM-Based Efficient Sketch Recognition

Tevfik Metin Sezgin and Randall Davis
MIT Computer Science and Artificial Intelligence Laboratory
77 Massachusetts Ave, 32-235
Cambridge MA 02139
{mtsezgin,davis}@csail.mit.edu

ABSTRACT

This paper shows how viewing sketching as an interactive process allows us to model and recognize sketches using Hidden Markov Models. With the increasing availability of tablet notebooks and pen-based PDAs, sketch based interaction has gained attention as a natural interaction modality. Current sketch recognition architectures treat sketches as images or a collection of strokes, rather than viewing sketching as an interactive and incremental process. We report results of a user study indicating that in certain domains people have preferred ways of drawing objects. We show how the consistent ordering of strokes, when present, can be used to perform sketch recognition efficiently. This novel approach enables us to have polynomial time algorithms for sketch recognition and segmentation, unlike conventional methods with their exponential complexity.

Keywords Sketch recognition, Enabling input technologies, Interpretation of user input, Intelligent user interfaces

1. INTRODUCTION

Sketches help us convey ideas, guide our thought process, and serve as documentation [2]. Most importantly, sketching is a natural input modality of increasing interest [9]. Recognizing their value, several authors have suggested sketch-based systems that use sketching as a natural input modality, emphasizing the user interfaces aspect [7, 6, 15, 10, 19, 23]. Complementing this work, others have suggested sketch recognition systems that put more emphasis on recognizing complex objects [4, 13, 12]. Our work is in the latter spirit and proposes a novel approach to symbolic sketch recognition that takes advantage of the incremental and interactive nature of sketching.

1.1 Terminology

By a sketch, we mean messy, informal hand-done drawings as in Fig. 1. Specifically we are interested in recognizing sketches of objects that can be described and recognized using structural methods, the class of sketches that has been the focus of the sketch recognition community [3, 4, 6, 13, 12]. We view sketching as an incremental process, defining a sketch as a sequence of strokes. Strokes are collected using a digitizing LCD tablet or a tablet computer that tracks both x, y position and the time t for each point. Note that we know the order in which the strokes are drawn, and as we describe below, stroke ordering is an important source of knowledge in recognition.

We use the term ***sketching style*** to refer to a user's preferred – although not necessarily conscious – stroke ordering when drawing an object. We characterize the sketch recognition process in terms of three tasks:

- ***Segmentation:*** The task of grouping strokes so that those constituting the same object end up in the same group. At this point it is not known what object the strokes form. For example, in Fig. 1, the correct segmentation gives us four groups of strokes.
- ***Classification:*** Classification is the task of determining which object each group of strokes represent. For Fig. 1, recognition would indicate that the first object in the sketch is a stick-figure.
- ***Labeling:*** Labeling is the task of assigning labels to components of a recognized object (i.e., the head, the torso, the legs and the arms in the stick-figure in Fig. 1).

1.2 The problem

Current sketching systems are indifferent to who is using the system, employing the same recognition routines for all users. But a user study we have conducted clearly indicates that although sketching styles may vary across users, people have consistent sketching styles. We believe there is considerable value in being able to capture these different styles and use them to aid recognition. The framework we introduce in this paper provides a mechanism for capturing an individual's preferred stroke ordering during sketching, and uses it for efficient sketch recognition. We show how viewing sketching as an incremental and interactive process provides extra leverage in sketch recognition.

The framework we introduce also provides efficient segmentation and classification. As noted in [12], treating sketches as images leads to recognition algorithms with exponential time complexities: for example, subgraph isomorphism based methods have exponential time complexities, while decision-tree based approaches have exponential storage requirements. If we assume having m object classes each object model with k components, a simple calculation shows that in the worst case, recognition of an object on a sketching surface with n strokes requires $\binom{n}{k}$ grouping operations and $k!$ constraint checking operations, yielding a total of $m\binom{n}{k}k!$ operations. In practice, the combinatorics get even worse

Extended from Sezgin-Davis IUI 2005.

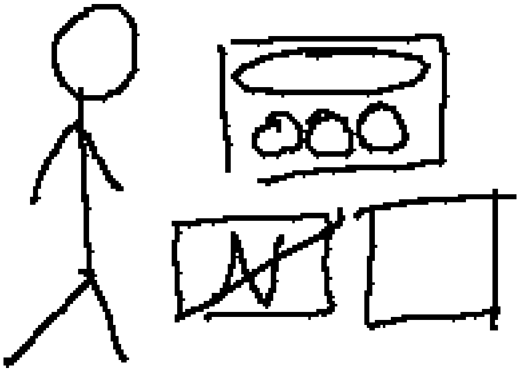

Figure 1: Example showing what we mean by a sketch. Note the messy, freehand nature of the drawings.

because sketches are inherently noisy, messy, and because transformation space algorithms – a class of efficient recognition algorithms [22] – are inapplicable due to the variation in relative sizes of object components (e.g., relative sizes of head and body in a stick-figure vary across figures). Exponential time and space requirements are unacceptable for interactive sketch recognition. With the same motivation, [12] suggests several heuristics that speed up recognition, but don't eliminate the exponential nature of the task. In this paper we show how treating sketching as an incremental and interactive process allows polynomial time recognition algorithms.

We next describe our approach to the sketch recognition problem; later sections give details on system implementation, results and evaluation. We conclude the paper with discussion of related and future work.

2. APPROACH

Our approach is motivated by static and dynamic characteristics of sketches and it is differentiated from work on images by the need to deal with the dynamic character of sketches.

2.1 Characteristics of Sketching

Sketches have a number of static properties (i.e., properties found in images, pictures, or scanned documents). Unlike formal drawings, they are are messy (e.g., Fig. 1), and are usually iconic (e.g., a human is often represented by a stick-figure icon). Sketches are often compositional; a house, for example, is formed by composing an isosceles triangle and a rectangle, with the triangle above the rectangle.

In addition to these static properties, we can view sketching as a dynamic process that is incremental, interactive, and highly stylized. By incremental we mean strokes are put on the sketching surface one at a time. Sketch recognition can be viewed as interactive, because there is two way communication, from the user to the computer in terms of the drawn strokes and the editing operations, and from the computer to the user in terms of the computer's interpretation and display of the strokes and editing operations. Finally, sketching is highly stylized: people have strong biases in the way they sketch, which in turn forms the basis for our approach to solving the problems described above.

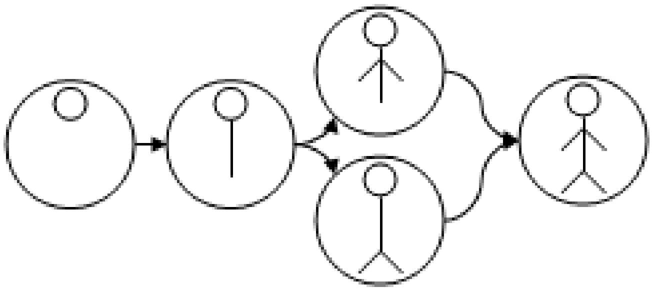

Figure 2: A sketching style diagram showing two ways of drawing stick-figures

2.2 User study

We ran a user study to assess the degree to which people have sketching styles, by which we mean the stoke order used when drawing an item. For example, if one starts drawing a stick-figure with the head, then draws the torso, the legs and the arms respectively, we regard this as a style different from the one where the arms are drawn before the legs (see Fig. 2). Our user study asked users to sketch various icons, diagrams and scenes from six domains. Example tasks included:

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

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

Fig.4 shows statistics on drawing orders. The maximum possible drawing orders for each object shows the theoretical upper-bound on the number of possible drawing orders. In theory, there are $n!$ ways of drawing an object with n subcomponents, but as seen here, only a few orders are actually preferred by the users. The table also shows the mean number drawing orders for all users, rounded to the nearest integer.

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

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

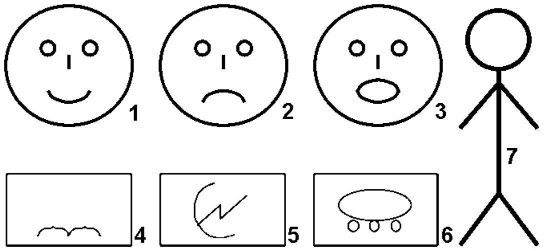

Figure 3: Examples of the figures that the users were asked to draw in the user study.

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

Figure 4: A summary of the statistics from our user study for a subset of the symbols drawn by the users (shown in Fig.3). Individual user statistics are not included due to space limitations. Note that 5!=120, 6!=720 and 8!=40320. Users drew 30 or more examples of each object.

- People sketch objects in a highly stylized fashion. In drawing the stick figure, for example, one of our subjects always started with the head and the torso, and finished with the arms or the legs (Fig.2).
- Individual sketching styles persist across sketches.
- Subjects preferred an order (e.g., left-to-right) when drawing symmetric objects (e.g., the two arms) or arrays of similar objects (e.g., three collinear circles).
- Enclosing shapes are usually drawn first (e.g., the outer circle in emoticons, or the enclosing rectangles in Fig. 3).

The user study confirmed our conjecture about the stylized nature of sketching. In order to capitalize on this structure we have used Hidden Markov Models (HMMs) to model different sketching styles. We next describe why we use HMMs, briefly review HMMs, and explain how we applied them to the problem at hand.

3. HMM-BASED RECOGNITION

3.1 The intuition

To give an intuitive explanation of why and how we use HMMs for recognition, consider an over-simplified scenario. Assume we have only two types of objects: *skip-audio-track* and *stop* symbols (Fig. 5), and assume the user always draws them using the same stroke ordering, indicated by the numbers. Our task is to recognize which of these objects is present in a given scene that is known to contain only a single instance of one of these objects. Suppose the user draws the *stop* symbol as shown in Fig. 5. Assuming we can reliably recognize the individual strokes as lines and tell whether they are horizontal (**H**), vertical (**V**), negatively/positively sloped (**N**, **P**), we can look at the order in which the user drew the lines and classify the input as a *stop* symbol if we see the [**V, H, V, H**] ordering, and as a *skip-audio-track* symbol for the [**V, P, N, V**] ordering.

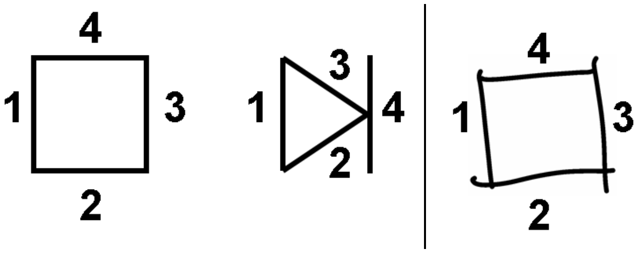

Figure 5: Symbols for *stop* and *skip-audio-track* (on the left), and a sketched *stop* symbol (right).

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

- **Support for multiple classes and drawing orders:** We should be able to recognize multiple instances of objects from many classes and accommodate multiple drawing orders.
- **Handling variations in drawing and encoding length:** The users should be able to draw freely, for example, they should be able to draw the *stop* symbol using three strokes instead of four (thus generating an encoding of the sketch with only three observations instead of four).
- **Probabilistic matching score:** The result matching an observation sequence against a model should reflect the likelihood of using that particular drawing order for drawing the object. This is required if we are to have a mathematically sound framework for combining the outputs of multiple matching operations for scenes with multiple objects, as we explain later.
- **Learning:** In practice, different drawing orders will have similar subsequences. Ideally the system should learn compact representations of drawing orders from labeled sketch examples. In addition, if a user starts the *stop* symbol with a vertical line 20% of the time, this fact should be used by recognition and training.

The requirements above imply the need for a systematic way of measuring how well observation subsequences match individual object models and a well-founded method for learning multiple drawing orders for objects. HMMs provide a well-founded, mathematically sound foundation for learning models of sequential patterns from training data and for testing how well a particular model matches a sequence of observations. Next we briefly review HMMs, then show how to combine the results from individual HMMs to perform recognition and segmentation.

3.2 Overview of HMMs

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

In the toy example discussed above, the symbols v_i correspond to the encodings of input strokes in terms of lines of different orientations. The states q_i can be thought of embedding the knowledge of having seen a particular observation sequence. The observation probabilities $B_j(v)$ give us the probability of observing each of the primitives given our current state. A captures the state transition dynamics.

Given an HMM $\lambda(A, B, \pi)$ and a sequence of observations $O = o_1, o_2, .., o_k$, we can efficiently determine how well each model λ accounts for the observations by computing $P(O|\lambda)$ using the Forward algorithm; compute the best sequence of HMM state transitions for generating O using the Viterbi algorithm;[1] and estimate HMM parameters A, B and π to maximize $P(O|\lambda)$ using the Baum-Welch algorithm.[2]

3.3 Modeling with HMMs

3.3.1 Encoding

Sketches must be encoded to generate observation sequences for recognition. We encode strokes using the Early Sketch Processing Toolkit described in [3] which converts strokes into geometric primitives. We encode the output of the toolkit to convert sketches into discrete observation sequences using a codebook of 13 symbols; four to encode lines: positively/negatively sloped, horizontal/vertical; three to encode ovals: circles, horizontal/vertical ovals; four to encode polylines with 2, 3, 4, and 5+ edges; one to encode complex approximations (i.e., mixture of curves and lines); and one symbol to denote two consecutive intersecting strokes. The choice of an encoding scheme is an important issue that can affect recognition accuracy. As we show, we obtained very promising results with this encoding.

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

3.3.2 Modeling with fixed input length HMMs

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

For isolated object recognition, we compute $P(O|\lambda_i)$ for each model λ_i using the Forward procedure with the observation sequence O generated by encoding the isolated object. λ_i with the highest likelihood gives us the object class. Unfortunately isolated object recognition requires the input sketch to be presegmented, which is usually not the case, and segmentation is itself a hard problem.

Interpretation of a complex scene requires generating hypotheses for the whole scene. That is, it requires segmenting the entire observation sequence into subsequences and assigning models to these segments so that all the strokes in the scene are accounted for. This requirement of accounting for all strokes implies that interpretations should be chosen so that they form a *globally coherent* interpretation for the whole scene.

The hypothesis generation should be efficient, so combinatoric approaches are ruled out. The fact that individual HMMs return probability values makes it easy to define the objective for this stage, namely, choose interpretations that maximize the probability corresponding to the entire scene. This is an optimization problem that we solve using dynamic programming implemented in the form of a shortest path problem.

The shortest path in a graph $G(V, E)$ that we generate gives us the segmentation. We then perform classification as described above. The graph $G(V, E)$ that we build for segmentation is distinct from the graphs that represent HMM topologies. Segmentation and recognition begins by building the graph $G(V, E)$ such that: V consists of $|O|$ vertices, one per observation, and a special vertex v_f denoting the end of observations. Let k be the input length for model λ_i. Starting at the beginning of the observation O, for each observation symbol O_s, we take a substring $O_{s,s+k}$ and compute the loglikelihood of this substring given the current model, $log(P(O_{s,s+k}|\lambda_i))$. We then add a directed edge from vertex v_s to vertex v_{s+k} in the graph with an associated cost of $|log(P(O_{s,s+k}|\lambda_i))|$. If the destination index $s + k$ exceeds the index of v_f, instead of trying to link v_s to v_{s+k}, we put a directed edge from v_s to the final node v_f. We set the weight of the edge to $|log(P(O_{s,|O|}|\lambda_i))|$. Here $O_{s,|O|}$ is the suffix of O starting at index s.[3] We complete the construction of G by repeating this operation for all models.

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

A nice feature of the graph-based approach is that while the shortest path in G gives us the most likely segmentation of the input, we can also compute the next k-best segmentations using a k-shortest path algorithm. This information can be used by another algorithm for dealing with ambiguities or by the user, as done in speech recognition systems

[1] We use this information in determining if a sequence of observations describe a complete object.

[2] This is how we learn probabilistic models of how the user draws objects from drawing examples.

[3] The ability to do recognition when the scene is not yet complete is a major challenge in recognition that most other systems sidestep by requiring the user implicitly or explicitly aid segmentation. Adding these special edges from v_s to v_f for $s + k > |O|$ allows our segmentation and recognition algorithms to work even if the user hasn't completed drawing the current object while preserving global consistency. This feature is a major strength of our approach. We leave out a detailed discussion and evaluation results on this feature due to lack of space.

with n-best lists.

3.3.3 *Modeling using HMMs with variable length training data*

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

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

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

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

3.4 Implementation

We used BNT [16] written in MATLAB as our main HMM engine. The graph construction, segmentation and recognition algorithms were implemented in Java and used probabilities computed by the HMM Toolkit.

Because sketching is incremental, we preferred the Bakis (left-to-right) HMM topology. This is done by initializing $a_{ij} = 0$ for $i > j$ for each model λ_i. B, π and the other entries in A are set to random values preserving stochastic properties. We used the maximum number of nodes in the sketching style diagrams obtained from our user study to set the number of states per HMM to 10.

4. EVALUATION

Evaluation of our work consists of two parts: Evaluation of the HMM-based recognition approach with real data, and a second experiment to compare the performance of our algorithm to a baseline method.

4.1 Evaluation of the HMM-based recognition

Our first experiment was aimed at measuring the suitability of our approach for sketch recognition and observing its behavior with test data containing clutter in the form of spurious strokes or unknown objects.

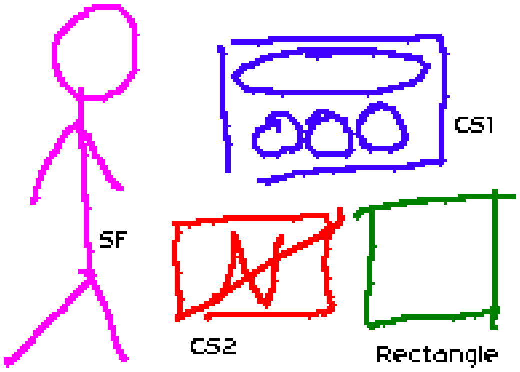

Figure 6: The output of our system for the test case shown in Fig.1 with drawing order CS2, CS1, stick fig., and rectangle.

We ran separate tests for the fixed and variable length input HMMs, in each experiment learning 10 object classes from the domains of geometric objects, military course of action diagrams, stick-figure diagrams, and mechanical engineering drawings. Training data was sketched using up to 6 styles with 10 examples per style to capture the variations in encoding for each style. The examples were manually segmented to obtain training data.

We compiled a test set – separate from the training data – consisting of a total 88 objects sketched using the sketching styles present in the the training data. The fixed input length HMM method had an accuracy of 96%. The variable input length method's accuracy was 81% without explicit ending states and 97% with them.

An example of our system's output for one of the test sketches using the second method is in Fig. 6. In this example, the course-of-action-diagram symbols, CS2, CS1 were drawn first, followed by the stick-figure and then the rectangle.

We also tested both methods with sketches including negative examples to measure their robustness in presence of unknown objects and spurious strokes. Two classes of negative examples were obtained by randomly inserting strokes selected from other sketches, and by simulating the effects of common low level recognition errors (e.g., classifying a stroke as a polyline with two segments instead of a line).

We observed that negative examples usually inhibit correct segmentation of objects drawn right before and after, but this effect remains bounded in a small neighborhood and other objects in the scene are correctly recognized. For example, for the fixed input length HMM method and 200 examples where we inserted one or more spurious strokes, the size of the neighborhood of misrecognition was 1 in 57% of the trials, 2 in 35% of the trials and 3 or more for the remaining 8%. In 200 trials where we simulated common low level errors at the single stroke level, 43.5% of the errors were within a neighborhood of 1, 22.5% were within 2, 3% were within 3 or more. In 7.5% of the cases, the low level error caused only the object containing the erroneous stroke to be misrecognized as two separate objects, thus not causing misrecognitions in preceding or following objects (*local* neighborhood). In the remaining cases, errors by the low level stroke processor did not result in objects being misrec-

	Misrecognition neighborhood		
	1	2	3+
Fixed length	57%	35%	8%
Variable length	69%	25%	6%

Figure 7: Effects of spurious strokes on recognition errors for fixed and variable input length HMMs.

	Misrecognition neighborhood				
	none	local	1	2	3+
Fixed length	23.5%	7.5%	43.5%	22.5%	3%
Variable length	28%	6%	44%	20%	2%

Figure 8: Effects of low level errors on recognition errors for fixed and variable input length HMMs.

ognized because the training data naturally contained examples where low level processing had failed.

The results for spurious strokes are tabulated in Fig. 7 for both recognition methods, and Fig. 8 shows the effects of low level errors. As seen in these tables, the variable length input model does slightly better. We believe this is because we have more training data per HMM in this model (our original motivation for introducing the variable input length method).

4.2 Running time comparison to a baseline method

We compared the performance of our system to a baseline method using feature-based pattern matching without stroke ordering information. In this experiment, we aimed to compare how the running time of our method and a feature-based method scaled with respect to the number of unrecognized objects in a scene. [4]

To serve as a baseline, we implemented a feature-based recognizer as used in [12, 4, 13].[5] The baseline method takes a structural object description as input and recognizes objects by assigning scene elements to model parts such that spatial relationships between scene elements agree with those specified by the model. Object are described in the object description language proposed by Hammond in [18]. Fig. 9 shows the object description for the stick-figure. The description lists the types and names of the components forming the object and the constraints that must be satisfied between them to declare a stick-figure instance present.

We also implemented a compiler that takes an object description and compiles it into a series of Java expert system shell (Jess) rules that collectively act as a recognizer. The compiler-generated rules enforce the constraints stated in the object description, ensure that each scene element is assigned to only one object model, and ensure labeling of line endpoints (as p1, p2) is consistent with the object model. Recognition is done by an engine that efficiently does many to many matching, avoiding duplicate computation by bookkeeping (see [17] for details).

```
(define Stick-Figure
 (components
   (Ellipse head)
   (Line body)
   (Line leftArm)
   (Line rightArm)
   (Line leftLeg)
   (Line rightLeg))
 (constraints
C0    (meets leftArm rightArm leftArm.p1 rightArm.p1)
C1    (meets leftLeg rightLeg leftLeg.p1 rightLeg.p1)
C2    (left leftArm     rightArm)
C3    (left leftLeg     rightLeg)
C4    (touches leftArm  body leftArm.p1)
C5    (touches rightArm body rightArm.p1)
C6    (meets leftLeg    body leftLeg.p1 body.p2)
C7    (meets rightLeg   body rightLeg.p1 body.p2)
C8    (touches body     head body.p1)))
```

Figure 9: Stick-figure and the object model.

The formulation of the baseline method as described above is symbolic and feature-based. It doesn't use training data, and is not built to be robust in the face of errors in the low level processing (e.g., a straight stroke incorrectly being broken into two collinear lines opposed to a single line). As a result, object recognition naturally fails if there are low level processing errors. To achieve a fair comparison, we ran the baseline system on input without low level errors. This ensures perfect recognition for the baseline system, and allows us to take measurements on the time needed for recognizing scenes with varying number of objects.

We ran the baseline system on scenes containing up to 5 objects (rectangles or stick-figures in this case). To avoid stroke orderings that are unusually ambiguous for the baseline method, we measured the mean recognition times for different drawing orders. We started the experiment with a scene containing a single rectangle and measured the average recognition time for different orders. Then, in an alternating manner, we added either a stick-figure or a rectangle to get a new scene and measured the average recognition time for different orderings again. We repeated the experiment with up to 5 objects (three rectangles, two stick-figures).

Fig. 10 shows the average times for increasing number of objects. The same figure also shows the running times for the HMM-based method, using hand-drawn examples of the same objects selected from the previously mentioned collection of 88 sketches. As seen in Fig. 10, the HMM-based method scales very well with increasing number of objects. This is because matching a model λ_i to an observation sequence of length T takes only $O(N^2T)$ operations, where N is the number of states in λ_i. This is linear in the number of observations. In fact, our method can process scenes with up to 80 objects in less than a minute. Performance of the baseline method gets worse as the number of objects increases, because primitives of the additional object parts act as distractors. The performance of the baseline method can conceivably be improved using elaborate segmentation, preprocessing or perceptual organization (e.g., [11]), but we still regard the above results as promising.

Both the baseline method and the HMM-based method were run on a Pentium III 933 MHZ machine with 512M of memory, running Windows XP. These results show how valuable drawing order information can be when users sketch in predictable orders.

4.3 Discussion

Our evaluation also revealed some weaknesses of our ap-

[4] Scalability is a must for being able to keep multiple hypotheses around for a larger portion of the sketch. This makes it possible for the recognition system to avoid commiting to interpretations immediately. This, in turn, allows maintaining hypotheses for a larger portion of the sketch and updating them when more data becomes available as it is the case in online sketch recognition.

[5] An HMM-based approach without the stroke ordering information would be inappropriate as a baseline because it would have very poor accuracy.

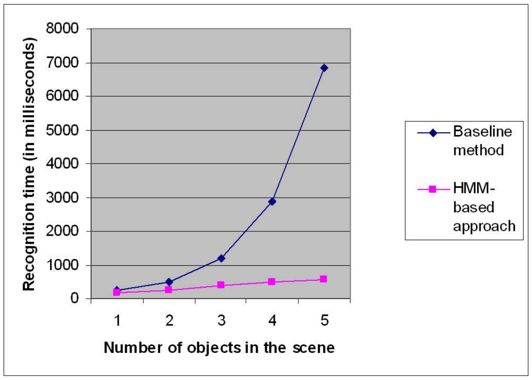

Figure 10: Running times for the baseline feature-based recognition system and our HMM-based recognition system.

proach. HMMs are suited for sequence analysis. For objects that produce single observations (objects drawn using a single stroke), building an HMM amounts to deriving the prior of a single observation and is not as useful in recognition. Our technique thus works best with objects drawn with multiple strokes, or are encoded to yield multiple observations.

Also the scheme we presented relies on having sketching styles. The system will not be appropriate for domains where there is no consistent sketching style. In that case, alternate object recognition methods, which are relatively more costly, can be used. Ideally the two methods would exchange information to aid one another for higher speed and accuracy.

Finally, because we use an orientation dependent encoding, we recognize objects in their canonical orientations. Orientation invariant recognition can be achieved by using rotation invariant encodings or by using synthetic training data obtained by rotating sketches in canonical orientations.

5. CONTRIBUTIONS

We have established through our user study that people have preferred stroke orderings for drawing objects. Although different people may have different preferences, individual preferences are persistent and agree across sketches.

Based on the user study, we suggested a sketch recognition framework fast enough to work in real-time. The most notable feature of our framework is that it allows fast, scalable segmentation and classification for sketches. Our algorithms have polynomial time complexity, unlike combinatoric structural methods with exponential complexities. Computing $P(O|\lambda_i)$ in our model takes $O(N^2T)$ operations, for an HMM λ_i with N states, and an observation O of length T.

One of the most attractive features of our system is that it complements model-based sketch recognition systems that don't make assumptions about the drawing order but search many possible segmentations for recognition. We use regularities in sketching to do fast recognition. We believe the two methods can work together: Given a scene, we can run the faster HMM-based system first. If the HMM-based system fails due to an unknown drawing order, the model-based recognition system can be invoked. If the model-based recognition system works successfully, the new drawing order can be added to the training set for that object class, enabling transparent learning of new styles without user intervention.

Finally, our system does not require that the user finish drawing the current object before it can be run (i.e., it doesn't need to be told that user is done sketching). It can be run after each stroke is added to the surface.

6. RELATED WORK

Several authors have suggested sketch-based systems that use sketching as a natural input modality and emphasize the user interfaces aspects [7, 6, 15, 10, 19, 23]. Here we limit our review to systems whose main focus is recognition.

Work in [4] and [13] describe grammar-based statistical approaches to sketch recognition. [5] presents a technique for generating bottom-up recognizers from object descriptions that perform exhaustive search. Our work naturally complements these systems.

In [12], Mahoney and Fromherz describe a system for sketch recognition from structural object descriptions using subgraph isomorphism for structural matching. Their approach is designed for recognizing sketches in scanned documents. They also provide a detailed discussion of the exponential time complexity of the structural sketch recognition task, which we tackle in this paper using drawing order information.

In [23], although sketch recognition is not their main focus, Cohen et al. describe a hybrid neural network HMM gesture recognizer for QuickSet to recognize isolated instances of symbols. Our approach handles scenes with multiple objects, without making segmentation assumptions.

HMMs have previously been used to do online handwriting recognition [20]. Here we show the suitability of HMMs for sketch recognition and use stroke-level observations as opposed to pixel-level chain-code-like representations used by the handwriting recognition community. Handwriting has allographic and neuro-biomechanical variability [21], which leads system designers to use chain-code-like representations. High recognition rates that we obtained suggest that although sketches have sequencing variability [21], they can be recognized using stroke level geometric descriptors. This is unlike handwriting recognition where using higher level descriptors, although studied in the 1960s, was abandoned for techniques that use pixel level chain-codes and quantization [20].

Our system also differs from numerous sketch recognition systems by its ability to do recognition with only polynomial time and space complexity, and by its utilization of drawing order for capturing and modeling user sketching styles.

7. FUTURE WORK

We are investigating how varying the number of states and graph topology affects system performance. We are exploring ways of inducing these properties from examples by applying Bayesian model merging techniques described in [8].

An interesting research question is to investigate how inter-object correlations can be captured using hierarchical HMMs. In fact, the machinery we described for learning variable length data using HMMs with explicit ending states can be used for this task. This model has the structure of a two level hierarchical HMM with uniform priors for the lower level nodes corresponding to objects. With training data from complete sketching sessions, domain specific transition probabilities for these nodes can be learned.

We are currently developing a new HMM architecture that will allow us to integrate our system with a model based sketch recognition architecture. This will allow us to collect data for a comprehensive study that will stress-test the system. Integrating the systems will be challenging because we want the systems to share hypotheses and help each other.

We believe the HMM framework is also appropriate for learning editing operations (e.g., deletion, selection) if corresponding observations are supplied during the learning phase. This can be especially useful in detecting common recognition errors. For example, if there is a chronic failure (i.e., consistent misrecognition) in recognizing stick-figures that occasionally requires the user to delete and redraw the head to achieve correct recognition, this can be captured by the HMM. It is an interesting research question to see to what degree this information can later be used by the recognition engine or by the programmer to modify recognition criteria.

Finally, we are planning to explore the characteristics of good encoding schemes. Ideally, a good encoding scheme should result in good recognition rates, but should also capture perceptual features of objects that humans seem to attend to. We believe features corresponding to perceptual phenomena will be essential for the approach mentioned above for detecting chronic recognition errors and generating explanations for why they occur in a human understandable language (i.e., “the recognition fails because the user has a tendency towards not drawing the head circular enough”, as opposed to ”because some obscure statistic of the stroke causes misrecognition” inspired by [14]).

8. SUMMARY

We showed how viewing sketching as an interactive process allows us to model and recognize sketches using Hidden Markov Models. We presented results of a user study indicating that in certain domains people have preferred ways of drawing objects. We illustrated how the consistent ordering of strokes naturally preferred by users can be exploited to build models for individual users and perform sketch recognition efficiently, without restricting the users to sketch in a certain way. Our approach enables us to have polynomial time algorithms for segmenting and recognizing complex scenes unlike conventional methods with their exponential complexity.

9. REFERENCES

[1] Rabiner L. R. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE, Vol. 77, no 2, February 1989.*

[2] Ullman D., Wood S. and Craig D. The Importance of Drawing in the Mechanical Design Process. *Computers and Graphics, vol 14-2, pp. 263-274 1990.*

[3] Sezgin T. M., Stahovich T. and Davis R. Sketch Based Interfaces: Early Processing for Sketch Understanding. *PUI'01.*

[4] Alvarado C., Oltmans M. and Davis R. *A Framework for Multi-Domain Sketch Recognition. AAAI Sketch Symposium 2002..*

[5] Sezgin T. M. Generating Domain Specific Sketch Recognizers from Object Descriptions. *Student Oxygen Workshop.* July 17, Gloucester MA, 2002.

[6] Gross M. D. and Do E. Ambiguous intentions: a paper-like interface for creative design. Proceedings of UIST 96, pp. 183-192, 1996.

[7] Landay A. J. and Myers B. A.. Sketching Interfaces: Toward More Human Interface Design. IEEE Computer, vol. 34, no. 3, March 2001.

[8] Stolcke A. and Omohundro S. Hidden Markov Model Induction by Bayesian Model Merging. *NIPS, San Mateo CA, 1993.*

[9] Stahovich T., Landay J. and Davis R., Chairs. *AAAI Sketch Understanding Symposium.* March 25-27, Stanford CA, 2002.

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

[11] Saund, E., Fleet, D., Mahoney, J., and Larner, D. Rough and Degraded Document Interpretation by Perceptual Organization, *Proc. SDIUT '2003, Maryland. pp. 121-133.*

[12] Mahoney V. J., and Fromherz M. P. J. Three main concerns in sketch recognition and an approach to addressing them. *AAAI Spring Symposium: Sketch Understanding.* March 25-27, Stanford CA, 2002.

[13] Shilman M., Pasula H., Russel S. and Newton R. Statistical Visual Language Models for Ink Parsing. *AAAI Spring Symposium: Sketch Understanding.* March 25-27, Stanford CA, 2002.

[14] Long, A. C. Jr., Landay J. A., and Rowe L. A. 'Those Look Similar!' Issues in Automating Gesture Design Advice. *PUI, November 15-16, 2001.*

[15] Forbus, K., Usher, J., and Chapman, V. Sketching for Military Courses of Action Diagrams. Proceedings of IUI03, January, 2003. Miami, Florida.

[16] Murphy K. Dynamic Bayesian Networks: Representation, Inference and Learning *Ph.D Thesis. UC Berkeley, Computer Science Division. July 2002.*

[17] Forgy C. L. RETE: A fast algorithm for the many pattern/many object pattern match problem *Artificial Intelligence 19(1):17-37, September 1982.*

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

[19] Hong I. J., Landay A. J., Long A. C., and Mankoff J. Sketch Recognizers from the End-User's, the Designer's, and the Programmer's Perspective. *AAAI Sketch Symp.* March 25-27, 2002.

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

[21] Schomaker L. From handwriting analysis to pen-computer applications IEEE Communication Engineering, 10(3), pp. 93-102, 1998

[22] Cass T. Polynomial-time geometric matching for object recognition International Journal of Computer Vision 21(1/2),37-61 (1997).

[23] Cohen, P.R., Johnston, M., McGee, D.R., Oviatt, S.L., Pittman, J., Smith, I., Chen, L., and Clow, J. QuickSet: Multimodal interaction for distributed applications. ACM MM97. pp:31-40.