EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2006)
Thomas Stahovich and Mario Costa Sousa (Editors)

An Efficient Graph-Based Symbol Recognizer

WeeSan Lee,[1] Levent Burak Kara,[2] and Thomas F. Stahovich[3]

[1]Department of Computer Science, University of California, Riverside, CA 92521
[2]Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213
[3]Mechanical Engineering Department, University of California, Riverside, CA 92521

Abstract

We describe a trainable symbol recognizer for pen-based user interfaces. Symbols are represented internally as attributed relational graphs that describe both the geometry and topology of the symbols. Symbol recognition reduces to the task of finding the definition symbol whose attributed relational graph best matches that of the unknown symbol. One challenge addressed in the current work is how to perform this graph matching in an efficient fashion so as to achieve interactive performance. We present four approximate graph matching techniques: Stochastic Matching, which is based on stochastic search; Error-driven Matching, which uses local matching errors to drive the solution to an optimal match; Greedy Matching, which uses greedy search; and Sort Matching, which relies on geometric information to accelerate the matching. Finally, we present promising results of initial user studies, and discuss the tradeoffs between the various matching techniques.

Categories and Subject Descriptors (according to ACM CCS): I.5.2 [Pattern Recognition]: Classifier Design and Evaluation

1. Introduction

Researchers have developed a variety of approaches for recognizing hand-drawn shapes and symbols. However, many of the current approaches have important limitations. For example, some methods are limited to single-stroke shapes drawn in preferred orientations [Rub91]. Others consider only aggregate properties of a shape and can confuse dissimilar shapes that have similar aggregate properties [FPJ02]. Other approaches require shapes to be drawn with a consistent pen stroke order [SD05].

Our work is aimed at overcoming some of these limitations. Our goal is to create an efficient, trainable, multi-stroke symbol recognizer that is insensitive to orientation, scaling, and drawing order. This is achieved via a graphical representation. Specifically, a symbol is represented with an attributed relational graph (ARG) describing its geometry and topology. The nodes in the graph represent the geometric primitives, and the edges represent the geometric relationships between them. Representing a symbol in terms of its topology allows us to achieve invariance to rotation and scaling, including non-uniform scaling. Because of the later capability, our approach is particularly tolerant of large variations in the shape of a hand-drawn symbol.

With our approach, symbol recognition reduces to the task of graph matching. During recognition, the ARG of the unknown symbol is matched against the ARG of each definition symbol to find the best match. The unknown is classified by whichever definition matches best. Graph matching, or sub-graph isomorphism [Ull76, DPZ01], is known to be NP-complete [GJ79]. Here, the problem is made more difficult because of noise. Noise comes from variations in how the symbols are drawn as well as from processing errors. For example, it is not uncommon for a symbol to have extra or missing geometric primitives, and thus extra or missing nodes in its ARG.

There has been considerable research in developing efficient graph matching techniques for a variety of applications [CFSV04]. Here we present and evaluate four new techniques specifically designed for recognizing hand-drawn shapes. These techniques are designed to be efficient enough for interactive performance, and to be tolerant of the noise inherent in hand-drawn symbols.

Our recognizer assumes that the individual symbols in a sketch have been located prior to recognition. In other work, we have developed sketch parsers for locating the symbols in a sketch [KS04, GKSS05].

The next section places this work in context by describing related work. This is followed by the details of our approach. Finally, results of a user study and conclusions are presented.

2. Related Work

Symbol recognition is an active area of research. An extensive overview of the literature can be found in [LVSM02]. Here, we present a representative sample of the literature.

Lee [Lee92] developed a graph-based recognizer in which the graph represents the precise geometry of the object. The approach is suitable for precisely drawn symbols with uniform scaling. For example, the approach has been used to recognize machine drawn symbols, symbols drawn using templates, and precise hand-drawn symbols. Lee's approach requires manual selection of key vertices during training.

Calhoun *et al.* [CSKK02] developed an approach in which the graph encodes topology, rather than geometry, so as to be more tolerant of variations in hand-drawn sketches. When learning definitions, thresholds are used to decide when a continuous property, such as intersection angle, should be included in the graph. If a property is included in the graph, it is represented by a single numerical value. In our work, attributes are described statistically, making our approach significantly more robust to pen stroke segmentation errors and drawing variations. Furthermore, to achieve interactive performance, Calhoun's approach requires the user to maintain a consistent drawing order. The system does have a mode that allows for variable drawing order. In that case, however, best-first search, which is computationally expensive, is used to do matching. The graph matching techniques we present are significantly more efficient. Likewise, Calhoun's approach requires the training examples to have a consistent drawing order, but this is not required for our approach.

In addition to symbol recognition, graph-based techniques have been used for a variety of other pattern recognition problems. Conte *et al.* [CFSV04] provides an extensive overview of graph matching techniques and their applications. According to the taxonomy presented there, our stochastic, error-driven, and greedy matching techniques can be considered approximate matching techniques based on continuous optimization.

Many existing approaches to symbol recognition rely on feature-based representations. Fonseca *et al.* [FPJ02] use features such as the smallest convex hull that can be circumscribed around the shape, the largest triangle that can be inscribed in the hull, and the largest quadrilateral that can be inscribed. Because their classification relies on aggregate features of the pen strokes, it might be difficult to differentiate between similar shapes. Rubine [Rub91] describes a trainable gesture recognizer designed for gesture-based interfaces. The recognizer is applicable only to single-stroke symbols, and is sensitive to the drawing direction and orientation. Pereira *et al.* [PBS*04] have extended Rubine's method to multi-stroke symbols. However, such symbols must be drawn with a consistent set of strokes. Additionally, they have developed a graph-based symbol recognizer, but it is not trainable. Matsakis [Mat99] describes a system for converting handwritten mathematical expressions into a machine-interpretable typesetting command language. Each symbol requires a multitude of training examples, where each example must to be preprocessed to eliminate variations in drawing directions and stroke orderings. However, the preprocessing makes their approach sensitive to rotations. Gennari *et al.* [GKSS05] describe a trainable recognizer that uses nine geometric features to construct concise probabilistic models of input symbols. The approach is suitable for multi-stroke symbols with arbitrary drawing orders and orientations. The features are an abstraction of the topology, thus is possible for shapes with different topologies to have the same features. Hse and Newton [HN04] developed a recognizer based on Zernike moments. The method is insensitive to rotation and uniform scaling. However, because the moments are essentially properties of a bitmap, the method is intolerant of non-uniform scaling.

In addition to graph-based and feature-based methods, researchers have also explored a variety of other representations and approaches. For example, Sezgin and Davis [SD05] present a technique based on hidden markov models. The approach requires shapes to be drawn with a consistent pen stroke ordering. Hammond and Davis [HD04] developed a recognizer that relies on hand-coded shape descriptions. Their representation is similar to ours in that both contain topological information. However, their descriptions are hand-coded while ours are learned from training examples. Gross' [Gro94] approach relies on a 3x3 grid inscribed in the symbol's bounding box. The sequence of grid cells visited by the pen distinguishes each symbol. Because of the coarse resolution of a 3x3 grid, this approach may not be able to handle symbols with small features. Kara and Stahovich [KS05] developed a recognizer based on a bitmap representation. One advantage of the approach is that it is tolerant of over-stroking and variations in line styles. However, the approach is sensitive to non-uniform scaling.

Parametric methods such as polygon, B-spline, and Bezier curve fitting techniques have also been considered in shape representation and classification [HC96, RVR02]. A benefit of these approaches is that there is no need to segment the pen stroke into geometric primitives such as lines and arcs. Additionally, since only a few parameters are needed for shape description, these methods are computationally efficient. Similar to the Rubine's method, however, these methods are primarily applicable to single-stroke symbols or gestural commands.

3. Representation

We represent a symbol with an attributed relational graph (ARG) describing its geometry and topology. The nodes in the graph represent the geometric primitives, and the edges represent the geometric relationships between them.

Each node is characterized by the type of the primitive – line or arc – and its relative length. The primitives are obtained from the raw pen strokes via a speed-based segmenter [Sta04]. The relative length of a primitive is defined as the ratio of its length (in pixels) to the total length of the primitives comprising the symbol. For example, each of the four line segments in a perfect square would have a relative length of 0.25. Defining length on a relative basis results in a scale-independent recognizer.

The edges in a graph represent the geometric relationships between the primitives. Each pair of primitives is characterized by the number of intersections between them; the relative locations of the intersections; and for lines, the angle of intersection. When extracting intersections from a sketch, a tolerance of 10% of the length of the segments is used to allow for cases in which an intersection was intended but one of the segments was a little too short. Intersection locations are measured relative to the lengths of the two primitives. For example, if the beginning of one line segment intersects the middle of another, the location is described by the coordinates (0%, 50%). The intersection angle is defined as the acute angle between two line segments. It is defined for both intersecting and non-intersecting line segments. Defining an intersection angle for non-intersecting segments allows the program to represent the topology of disconnected symbols, such as the dashpot in Figure 5. Intersection angle is not defined for an intersection between an arc and another segment.

Figure 1 shows an example of an ARG for an ideal square. Each side of the square has a relative length of 0.25 and intersects two other sides with an intersection angle of 90°. Because of the drawing directions used, all intersections are located at the end of one segment and the beginning of another.

A definition for a symbol is created by constructing an "average" ARG from a set of training examples. (Additional details of the training process are described in Section 6.) Each node in the average ARG is assigned the primitive type that occurred most frequently for that node in the training data. The number of intersections assigned to a pair of primitives is determined in an analogous fashion. A pair of primitives is assigned two intersections if at least 70% of the examples had two. If less than 70% had two intersections, but there was at least one intersection 70% of the time, the pair is assigned one. Otherwise, the pair is assigned zero intersections. The remaining properties of the ARG – relative length, intersection angle, and intersection location – are continuous valued properties. These are characterized by the means and standard deviations of the values from the training examples.

4. Measuring Similarity

During recognition, it is necessary to compare the ARG of the unknown symbol to the ARG of each definition symbol to find the best match. The unknown is classified by whichever definition matches best. The match between an unknown and a definition is quantified in terms of a dissimilarity score, which is computed using an ensemble of error metrics. These metrics consider both the intrinsic properties of the geometric primitives and the relationships between them. The former are encoded in the nodes of the ARG, the latter in the edges.

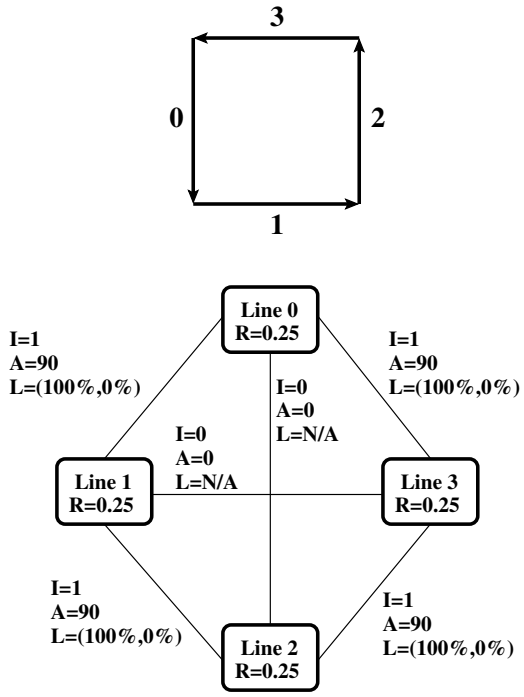

Figure 1: *Top: An ideal square drawn with a single, counterclockwise pen stroke. Arrows show the direction of drawing. Bottom: The corresponding ARG. I = number of intersections, A = intersection angle, L = intersection location, R = relative length.*

Error Metrics (E_i)	**Weight** (w_i)
E_1: Primitive count error	20%
E_2: Primitive type error	20%
E_3: Relative length error	20%
E_4: Number of intersections error	15%
E_5: Intersection angle error	15%
E_6: Intersection location error	10%

Table 1: *Error metrics and corresponding weights.*

Table 1 lists our six error metrics and the weights applied to them when computing the dissimilarity score. The weights, which are based on empirical studies, reflect the relative importance of the various error metrics for discriminating between symbols. For the purposes of recognition, the dissimilarity score is converted to a *Similarity Score* as follows:

$$Similarity\ Score = 1 - \sum_{i=1}^{6} w_i E_i \qquad (1)$$

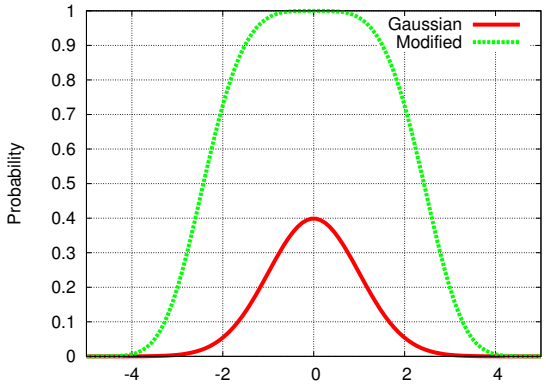

Figure 2: *Gaussian Probability Density Function and Modified Probability Density Function for* $\mu = 0$ *and* $\sigma = 1$.

where the E_i are the error metrics, and the w_i are the weights listed in Table 1.

The error metrics for relative length, intersection angle, and intersection location involve comparing properties of the unknown to distributions of those properties encoded in a definition. For example, it is necessary to compare the relative length of each primitive in the unknown to the mean and standard deviation of the relative length of the corresponding primitive in the definition. Ordinarily, this is done with a Gaussian probability density function. As an alternative, we have developed a modified probability density function (MPDF) that is better suited to our recognition task:

$$P(x) = exp[-\frac{1}{50.0} \cdot \frac{(x-\mu)^4}{\sigma^4}] \tag{2}$$

Here μ and σ are the mean and standard deviation of the observed features learned from the training examples. This function was designed empirically such that its top is flatter than the Gaussian probability density function with the same μ and σ. This makes it easier to detect matches that are in the "vicinity." Additionally, we have found that the Gaussian distribution dies off too quickly towards its tails, which decreases its usefulness for recognition. For comparison, Figure 2 shows both the Gaussian probability density function and our modified probability density function for $\mu = 0$ and $\sigma = 1$.

The six error metrics used for computing the similarity score are described in the following sections. Here we use the term "unknown" to refer to the symbol to be recognized, or equivalently, the ARG of that symbol. Likewise, the term "definition" refers to the ARG of a definition symbol. Note also that each metric is normalized to the range [0, 1] so that the weights in Table 1 have predictable influences.

4.1. Primitive Count Error

This metric compares the number of nodes in the unknown to the number in the definition. The error is defined as:

$$E_1 = min(1.0, \frac{|N_U - N_D|}{N_{min}}) \tag{3}$$

where N_U and N_D are the numbers of nodes in the unknown and the definition ARGs, respectively, and N_{min} is the minimum of N_U and N_D.

4.2. Primitive Type Error

This error metric accounts for differences between the primitive types of the corresponding nodes in the two ARGs. The error is defined as:

$$E_2 = \frac{N_{min} - \sum_{i=1}^{N_{min}} \delta(Type(U_i), Type(D_i))}{N_{min}} \tag{4}$$

where U_i is a node from the unknown, D_i is the corresponding node from the definition, $Type(X)$ is a function that returns the primitive type (arc or line) of node X, and $\delta(p,q)$ is one when $p = q$, and zero otherwise.

4.3. Relative Length Error

This error metric compares the relative lengths of the primitives of the unknown to those of the definition. Corresponding primitives should have similar relative lengths. If not, an error is assigned. Here, similarity is measured using the MPDF defined in Equation 2. The error is computed as:

$$E_3 = \frac{\sum_{i=1}^{N_{min}} [1 - P(U_R^i)]}{N_{min}} \tag{5}$$

where U_R^i represents the relative length encoded in the i^{th} node of the unknown ARG. $P(x)$ is evaluated using the mean and standard deviation from the corresponding node in the definition.

4.4. Number of Intersections Error

This error metric compares the intersections of the unknown with those of the definition. A pair of primitives in the unknown should have the same number of intersections as the corresponding pair in the definition. If not, an error is assigned. The total error is computed as:

$$E_4' = \frac{\sum_{i=1}^{N_{min}} \sum_{j=i+1}^{N_{min}} |I(U_i,U_j) - I(D_i,D_j)|}{min(M_U, M_D)} \tag{6}$$

where $I(X,Y)$ returns the number of intersections between the primitives in nodes X and Y, and M_U and M_D are the numbers of edges in the unknown and definition ARGs, respectively. A pair of primitives can intersect as many as two times. E_4' thus has a range of [0, 2]. So that all error metrics have the same range of [0, 1], the value of E_4' is "squashed" with the following function:

$$S(x) = \frac{1}{1+exp[6(1-x)]} \tag{7}$$

As a result, the "Number of Intersections Error" is defined as:

$$E_4 = S(E_4') \tag{8}$$

4.5. Intersection Angle Error

This error metric compares the intersection angles of the unknown with those of the definition. The intersection angle of a pair of lines in the unknown should be similar to that of the corresponding pair of lines in the definition. (Intersection angle is defined only for pairs of lines.) If not, an error is assigned. Here, similarity is again measured using the MPDF defined in Equation 2. The total error is computed as:

$$E_5 = \frac{\sum_{i=1}^{N_{min}} \sum_{j=i+1}^{N_{min}} [1-P(A_{ij})]}{\sum_{i=1}^{N_{min}} \sum_{j=i+1}^{N_{min}} T(Line(U_i,U_j),Line(D_i,D_j))} \tag{9}$$

where A_{ij} is the angle at which the primitive from node i of the unknown intersects the primitive from node j of the unknown. $P(A_{ij})$ is evaluated using the mean and standard deviation from the corresponding pair of primitives from the definition. Note that if the two primitives are not lines, A_{ij} is undefined and $P(A_{ij})$ is taken to be one. $Line(X,Y)$ is one when the nodes X and Y are both lines, and zero otherwise. $T(p,q)$ equals one if p and q are both one, and zero otherwise (i.e., T is the logical "and" operator). The numerator normalizes the error by the total number of intersections.

4.6. Intersection Location Error

This error metric compares the intersection locations of the unknown with those of the definition. The locations of the intersections between a pair of primitives from the unknown should be similar to those of the corresponding pair of primitives in the definition. If not, an error is assigned. Here, similarity is again measured using the MPDF defined in Equation 2. Because intersection location is defined by two coordinates, the MPDF is applied twice for each intersection. The total error is computed as:

$$E_6 = \frac{\sum_{i=1}^{N_{min}} \sum_{j=i+1}^{N_{min}} \sum_{k=1}^{I(D_i,D_j)} ([1-P(L_i^k)]+[1-P(L_j^k)])}{\sum_{i=1}^{N_{min}} \sum_{j=i+1}^{N_{min}} 2 \cdot I(D_i,D_j)} \tag{10}$$

where (L_i^k, L_j^k) is the coordinates of the k^{th} intersection between the primitives from nodes i and j of the unknown. $I(D_i,D_j)$ is the number of intersections between the primitives from nodes i and j of the definition. In cases where a pair of primitives intersect in the unknown but not in the definition, or vice versa, both $P(L_i^k)$ and $P(L_j^k)$ are set to zero.

5. Graph Matching

The previous section described how to compute the similarity between two graphs. This assumed that each node in the unknown ARG was assigned to a specific node in the definition ARG. This section describes how these assignments are obtained. This is a graph matching, or graph isomorphism problem. If the user always draws each symbol with a consistent number of segments and a consistent drawing order, the graph matching problem is trivial. In that case, drawing order would directly provide the correct node-pair assignments. In practice, however, users do not always maintain a consistent drawing order. Furthermore, the problem is made more difficult because of noise. Noise comes from variations in how the symbols are drawn as well as from processing errors. For example, it is not uncommon for there to be extra or missing nodes in the unknown (i.e., extra or missing geometric primitives). Likewise, a segment that was intended to be a line can be misinterpreted, either through ambiguity or processing errors, as an arc, or vice versa.

We have developed four efficient, approximate matching techniques to find the best match between two ARGs. These are: Stochastic Matching, Error-driven Matching, Greedy Matching, and Sort Matching. The first three methods are based on search. The fourth method avoids search by assuming a consistent orientation.

The search-based methods make initial node-pair assignments based on drawing order. Assignments are then swapped until the best match is obtained. The quality of the match at each iteration is determined using the similarity score defined in the previous section. Our three search-based approaches differ in the way they select the assignments to swap at each iteration.

If the two graphs being matched do not have the same number of nodes, the smaller one is "padded" with empty nodes. This ensures that every node in one graph has a match with a unique node in the other, and hence that every node is considered by the swapping process. When evaluating the error metrics, a pairing with an empty node produces the maximum possible local error. For example, the addition of empty nodes does not reduce the primitive count error, E_1.

Figure 3 illustrates the typical search-based process. For ease of explanation, the figure shows hypothetical symbols rather than ARGs. Finding the correct node-pair assignments is equivalent to finding the correct assignment of the segments of the unknown to the segments of the definition. Here, the segments of the definition symbol are numbered

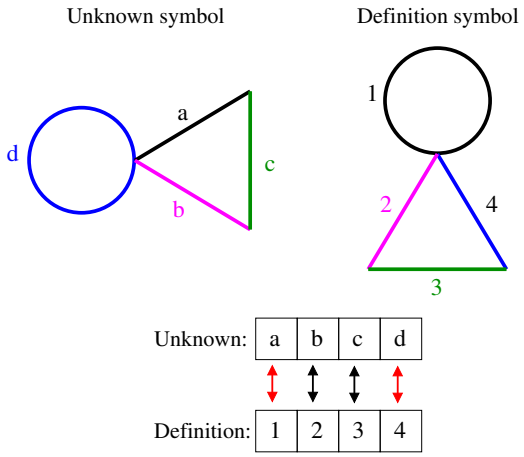

Figure 3: *Graph matching: assignments **b**-**2** and **c**-**3** are correct, while **a**-**1** and **d**-**4** are not.*

according to a typical drawing order. Likewise, the segments of the unknown are labeled with letters indicating the order in which they were actually drawn. Based on drawing order, segment **a** of the unknown is initially assigned to segment **1** of the definition, **b** is assigned to **2**, and so on. It is clear that assignments **b**-**2** and **c**-**3** are correct, while **a**-**1** and **d**-**4** are not. Swapping the latter to produce the assignments **d**-**1** and **a**-**4** is what is needed. The success of this swap can be measured by the resulting increase in the similarity score.

The following sections describe our four matching techniques in detail.

5.1. Stochastic Matching

This approach is based on stochastic search. To begin, the initial node-pair assignments are saved as the current best. Then, three node-pair assignments, which we will call A, B, and C, are randomly selected. A and B are swapped producing assignments A' and B'. B' is then swapped with C. If the new similarity score is better than the current best score, the new assignments are saved as the new current best. This process is repeated 100 times, and the current best node-pair assignments are returned as the best match. As this method is applied for a fixed number of iterations, the only cost that varies with problem size is the cost of evaluating the similarity score. This cost is $O(n^2)$, where n is the number of nodes.

5.2. Error-Driven Matching

With this approach, a local matching error determines the probability that a node-pair assignment will be selected to be swapped. For example, if a node from the unknown was a line primitive, and the corresponding node from the definition was an arc, there would be a relatively high local matching error, and correspondingly high probability that the node-pair would be selected for swapping. The local matching error of a node-pair is defined as the portion of the dissimilarity score related to that node-pair. This includes all intersection angle, intersection number, and intersection location errors involving the primitives in that node-pair. Likewise, the local error also includes segment type and relative length errors.

At each iteration, the local error of each node-pair is computed and selection probabilities are assigned. Based on these probabilities, two node-pairs are selected and swapped. If the similarity score improves, the new assignments are kept. Otherwise, the swap is rejected. This continues until there are 20 consecutive iterations with no improvement, or until 100 iterations have been performed. The computational complexity of this approach is similar to that of the Stochastic Matching approach, but this approach typically takes fewer iterations, and thus has lower cost.

5.3. Greedy Matching

This approach uses greedy search to find good node-pair assignments. The program first considers the best assignment for the first node of the unknown. If there are n nodes, the program considers all $n-1$ cases in which the first node-pair is swapped with another. Whichever assignment produces the best similarity score is selected for the first node, and this node-pair is removed from further consideration. This is repeated for the second node-pair and so on. In all, $O(n^2)$ sets of node-pair assignments are considered. For symbols with less than about 10 nodes, this approach is less expensive than the previous two.

5.4. Sort Matching

This approach does not rely on search. Instead, the nodes are sorted based on the locations of their primitives. Each line segment is characterized by its minimum x and y-coordinates. Each arc is characterized by the coordinates of its center. The primitives are then sorted in ascending order of their x-values. Ties are broken using the y-values. The sorted order of the nodes determines the node-pair assignments.

This approach is useful only when the drawing orientation is fixed. Likewise, variations in drawing can result in different sorted orders. Nevertheless, as Section 7 describes, the approach often works reasonably well in practice. Additionally, because this approach is particularly efficient, it is suitable for devices with little computational power, such as PDAs.

6. Training

The recognizer is trained by providing a set of training examples for each symbol class. As described in Section 3, the program constructs an "average" ARG for each class. This entails another graph matching problem. To learn a definition, the program must match the ARGs of the various training examples to one another. This task is different from

the previous matching problem because a similarity score cannot yet be computed. For example, the primitive type error cannot yet be determined because the expected primitive type of each node is yet to be determined.

We have explored two solutions to this problem. The first is to require the training examples to be drawn with a consistent drawing order. In this case, the matching problem is avoided as the drawing order uniquely identifies the nodes in an ARG. The second approach requires the user to draw symbols with a consistent orientation. In this case, geometric information is used for the matching. The training examples are scaled to have unit bounding boxes. The scaled symbols are then overlayed on top of one another. Finally, geometric proximity of the primitives is used to determine correspondence of the nodes. In particular, when two symbols are overlayed, each line or arc in one symbol is matched to the nearest line or arc in the other.

7. Results

We conducted a user study to evaluate the performance of our four matching techniques. The study involved nine participants. Each was asked to provide 15 examples of each of the 23 symbol classes shown in Figure 4. Data was collected using a Tablet PC. Figure 5 shows typical examples of the symbols drawn by the study participants.

	Stochastic	Error-driven	Greedy	Sort
Time (ms)	67.8	48.3	13.4	2.0
Top 1 (%)	93.7	93.5	92.3	78.5
Top 3 (%)	97.9	98.8	96.7	93.0

Table 2: *Results of user study: average time to classify a symbol and the top-one and top-three accuracies.*

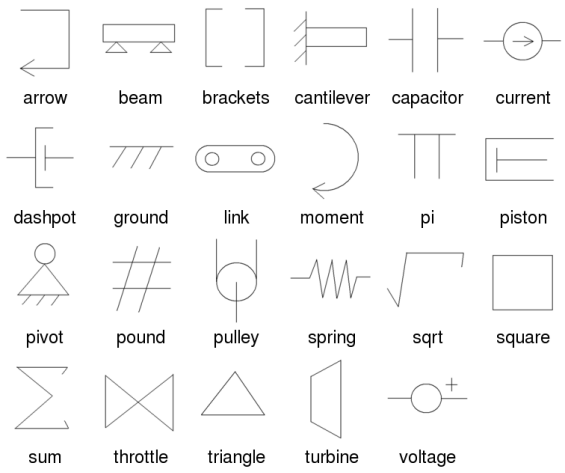

Figure 4: *Symbols used in the user study.*

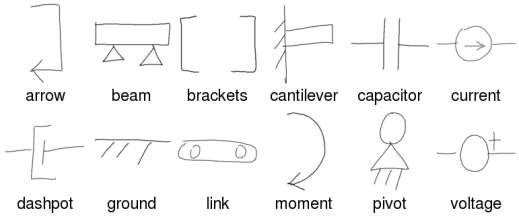

Figure 5: *Typical examples of symbols drawn in the user study.*

After the data was collected, recognition accuracy was determined offline. The participants received no feedback about the program's performance. To provide a good test of the matching ability of the four methods, the drawing order of each example was randomized after the data was collected. The system was tested in a user-dependent setting. For each experiment, the training and testing data were selected from a particular user. The results were then averaged across all users. Table 2 shows the recognition results when 14 examples were used for training. The training examples were randomly selected from the 15 examples provided by the particular user. This was repeated 10 times, using a cross-validation approach. The results in Table 2 are thus an average across 9 participants and 10 iterations of cross-validation.

The Stochastic, Error-driven, and Greedy methods all achieved similar performance of about 92% for top-one accuracy, and about 97% for top-three accuracy. Top-one accuracy is the rate at which the symbols were correctly classified. Top-three accuracy is the rate at which the correct class was one of the three highest ranked classes. The Error-driven approach took about 28% less time to recognize a symbol than the Stochastic approach. This is a result of using local matching error to guide the search to the best match. The Greedy approach worked well and took about 80% less time to recognize a symbol than the Stochastic approach. We expected that this approach would suffer from local maxima, but the results suggested otherwise. The Sort method achieved less accuracy than the other methods, but was significantly faster. This method worked reasonably well in these experiments because the participants drew the examples with a consistent orientation.

8. Conclusion

We have presented a trainable symbol recognizer for pen-based user interfaces. Symbols are represented internally as attributed relational graphs that describe both the geometry and topology of the symbols. Symbol recognition reduces to the task of finding the definition symbol whose attributed relational graph best matches that of the unknown symbol. One challenge addressed in the current work is how to perform this graph matching in an efficient fashion so as to achieve interactive performance. We presented four approximate graph matching techniques: Stochastic Matching,

which is based on stochastic search; Error-driven Matching, which uses local matching errors to drive the solution to an optimal match; Greedy Matching, which uses greedy search; and Sort Matching, which relies on geometric information to accelerate the matching.

Our initial experiments provided promising results. The Stochastic, Error-driven, and Greedy graph matching techniques all achieved at least 92% accuracy in a user-dependent setting with only 14 training examples. The top-three accuracy under the same conditions was 97%. While all three methods achieved similar accuracy, the Greedy approach was significantly faster than the others. The Sort Matching technique is less accurate than the others, and requires consistent drawing orientation. However, this technique is an order of magnitude faster than the others.

While more testing is needed to understand all of the tradeoffs between these approaches, our initial results suggest that Greedy Matching provides the best combination of speed and accuracy. However, when computational resources are constrained, such as with a PDA, Sort Matching is a good solution.

References

[CFSV04] CONTE D., FOGGIA P., SANSONE C., VENTO M.: Thirty years of graph matching in pattern recognition. *IJPRAI 18*, 3 (2004), 265–298.

[CSKK02] CALHOUN C., STAHOVICH T. F., KURTOGLU T., KARA L. B.: Recognizing multi-stroke symbols. In *AAAI Spring Symposium on Sketch Understanding* (2002), pp. 15–23.

[DPZ01] DICKINSON S., PELILLO M., ZABIH R.: Introduction to the special section on graph algorithms in computer science. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 10 (2001), 1049–1052.

[FPJ02] FONSECA M. J., PIMENTEL C., JORGE J. A.: CALI- an online scribble recognizer for calligraphic interfaces. In *AAAI Spring Symposium on Sketch Understanding* (2002), pp. 51–58.

[GJ79] GAREY M., JOHNSON D.: *Computers and Intractability: A guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

[GKSS05] GENNARI L., KARA L. B., STAHOVICH T. F., SHIMADA K.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4 (2005), 547–562.

[Gro94] GROSS M. D.: Recognizing and interpreting diagrams in design. In *ACM Conference on Advanced Visual Interfaces.* (1994), pp. 88–94.

[HC96] HUANG Z., COHEN F.: Affine-invariant b-spline moments for curve matching. *IEEE Transactions on Image Processing. 5*, 10 (1996), 1473–1480.

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

[HN04] HSE H., NEWTON A. R.: Sketched symbol recognition using zernike moments. *icpr 01* (2004), 367–370.

[KS04] KARA L. B., STAHOVICH T. F.: Hierarchical parsing and recognition of hand-sketched diagrams. In *UIST* (2004), pp. 13–22.

[KS05] KARA L. B., STAHOVICH T. F.: An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics 29*, 4 (2005), 501–517.

[Lee92] LEE S.-W.: Recognizing hand-drawn electrical circuit symbols with attributed graph matching. In *Structured Document Image Analysis*, Baird H. S., Bunke H., Yamamoto K., (Eds.). Springer-Verlag, 1992, pp. 340–358.

[LVSM02] LLADÓS J., VALVENY E., SÁNCHEZ G., MARTÍ E.: Symbol recognition: Current advances and perspectives. In *GREC '01: Selected Papers from the Fourth International Workshop on Graphics Recognition Algorithms and Applications* (London, UK, 2002), Springer-Verlag, pp. 104–127.

[Mat99] MATSAKIS N. E.: *Recognition of Handwritten Mathematical Expressions*. Master thesis, MIT, 1999.

[PBS*04] PEREIRA J. P., BRANCO V. A., SILVA N. F., CARDOSO T. D., FERREIRA F. N.: Cascading recognizers for ambiguous calligraphic interaction. pp. 63–72.

[Rub91] RUBINE D.: Specifying gestures by example. *Computer Graphics 25* (1991), 329–337.

[RVR02] RAYMAEKERS C., VANSICHEM G., REETH F. V.: Improving sketching by utilizing haptic feedback. In *AAAI Spring Symposium on Sketch Understanding* (2002), AAAI Press, pp. 113–117.

[SD05] SEZGIN T. M., DAVIS R.: HMM-based efficient sketch recognition. In *International Conference on Intelligent User Interfaces (IUI'05).* (New York, 2005).

[Sta04] STAHOVICH T. F.: Segmentation of pen strokes using pen speed. *AAAI 2004 Fall Symposium: Making Pen-Based Interaction Intelligent and Natural* (2004).

[Ull76] ULLMANN J. R.: An algorithm for subgraph isomorphism. *Journal of the ACM 23*, 1 (1976), 31–42.