EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2010)
M. Alexa and E. Do (Editors)

Computing Confidence Values for Geometric Constraints for use in Sketch Recognition

J. Johnston & T. Hammond

Department of Computer Science and Engineering
Texas A&M University
College Station, TX 77840
jbjohns & hammond@cse.tamu.edu

Abstract

Geometric constraints are used by many sketch recognition systems to perform high-level assembly of components of a sketch into semantic structures. However, with a few notable exceptions, most of the current recognition systems do not have constraints that use real-valued notions of confidence. We discuss methods for assigning confidence values to different kinds of constraints. We show how these confidence values equate to user perception, how they can be used to balance speed and accuracy in recognition algorithms, and how they can be used to assign confidence values to the high-level shapes they are used to construct. We use these constraints to extend the LADDER shape definition language in a system that recognizes 5,900 hand-drawn examples of 485 different military course-of-action diagrams at an accuracy of 89.9%.

1. Introduction

Sketch recognition lies at the intersection of human-computer interaction and machine learning/artificial intelligence. The power and appeal of sketch recognition lies in the belief that using a pen as an input modality, and communicating with the computer via sketch, can be more practical, efficient, and effective, at times, than typical usage of the mouse and keyboard.

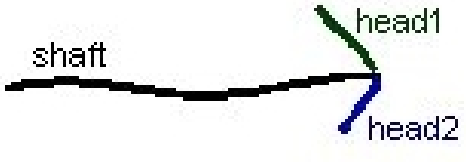
The work presented here focuses on high-level sketch recognition. High-level sketch recognizers take low-level primitive shapes and compose them into more complex constructions. Primitive shapes include atomic structures such as lines, curves, arcs, and circles, and are found through a combination of corner finding/stroke segmentation and primitive recognition techniques [SSD01, KK06, WEH08, PH08]. High-level shapes are usually defined using a set of geometric constraints placed on their low-level subparts. For example, an arrow might be composed of three lines, with constraints placed on them such that when the constraints are satisfied, the shape of an arrow is readily apparent. Figure 1(a) gives a set of constraints placed on three line primitives, called `shaft`, `head1`, and `head2`, respectively, that might be used to denote an arrow shape. Figure 1(b) shows a sketch that might be interpreted to satisfy the constraints, resulting in recognition of the three lines as an arrow.

Our work builds on LADDER [HD05]. We re-implement a subset of the constraints devised for the LADDER sketch recognition system, but do so in a way that provides confidence measures on each constraint. By incorporating confidences into the constraints, rather than using simple binary decisions, we allow for arbitrary levels of uncertainty in our recognition algorithms. Constraint confidence values afford us a method for assigning a confidence value to the end-product high-level shape that is a result of recognition. We also add the ability to customize the thresholds used by the constraints, giving the user of the constraints/visual language greater flexibility when creating definitions to have their sketches recognized by the computer. Most importantly, confidence values provide the ability to handle multiple interpretations more effectively. A sketch is inherently imperfect, and any sketch will have noise. As more and more shapes are added to a domain, multiple interpretations will become prevalent as more than one interpretation could be correct given appropriate context. However, to make the best classification decision possible, we would like a measure of how correct is any particular interpretation. Confidence mea-

DOI: 10.2312/SBM/SBM10/071-078

```
shape line shaft
shape line head1
shape line head2
shaft.end2 coincident head1.end1
shaft.end2 coincident head2.end1
head1 acute-meet head2
head1 same-size head2
shaft longer head1
```

(a) Constraints placed on the low-level lines of an arrow shape



(b) Example sketched arrow that might meet the constraints in (a)

Figure 1: *In high-level shape recognition systems, primitive shapes, such as lines, are composed into high-level shapes, such as arrows, using a shape definition that places constraints on the different subparts. The notation* `shaft.end2` *in (a) denotes one of the endpoints of line* `shaft`.

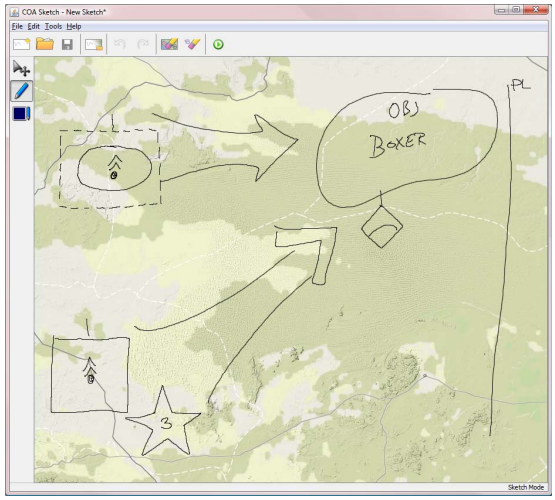
sures give us a value of correctness. Having a probabilistic measure for shape confidence allow us to use standard techniques to automatically combine shape confidence information with contextual information and prior probabilities as in [AD04] and [SPRN02].

A front-end interface for this work used in the context of recognizing military course-of-action diagrams has been implemented [HCD*10]. Figure 2 shows an example of a user's interaction with the COA application. In Figure 2(a), the user has drawn a COA sketch. Figure 2(b) shows the results of recognition after processing by the underlying sketch recognition algorithms, including those that use our constraints.

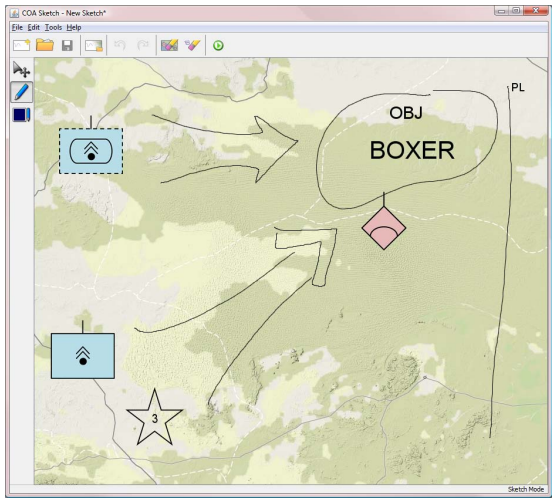
2. Related Work

Landay and Myers used constraints in their SILK application to compose strokes together [LM95]. The recognized high-level shapes were assigned semantic roles in a GUI, such as buttons and scroll bars. This allowed SILK's users to quickly sketch out and design a user interface using rapid prototyping methods, without having to code up an interface for testing. However, they only had a few constraints that captured the ideas of containment, nearness, and whether or not a sequence of objects drawn together were of the same type (for grouping radio button widgets). Our work, based on LADDER, offers a much wider variety of constraints, and computes confidence values–which SILK does not provide.

Shilman et al. improved on the recognition capabilities of SILK, improved the number of constraints provided, and



(a) COA Sketch before recognition



(b) COA Sketch after recognition

Figure 2: *An application has been built that allows a user to hand-draw course of action (COA) diagrams and have them dynamically recognized. Figure (a) shows an actual user's drawing on the application interface. Figure (b) shows the results of recognition after processing by the underlying sketch recognition algorithms, including those that use our constraints.*

added a statistical model for improving recognition by offering disambiguation capabilities [SPRN02]. Their work greatly improved the recognition capabilities of constraint-based recognition systems, and their use of statistical models provided confidence values. Like SILK, their set of constraints is fairly limited, but they were among the first practitioners to use statistical distributions to model constraints.

Lee, Kara, and Stahovich describe a method for using attributed relational graphs (ARGs) to represent sketched symbols [LKS07]. Nodes in the graphs represent primitive shapes, and edges between the nodes represent constraints between the shapes. Statistical distributions are estimated over the ARGs (μ and σ) using training data, and are later used to compare new sketch graphs to training samples and make classification decisions.

Alvarado [AD04] uses the same set of constraints and language semantics as those that our work is based on, but each with different high-level recognition algorithms. Alvarado uses dynamic Bayesian networks (DBNs) to model strokes and shapes as they are input to the system, computing posterior probabilities for each shape given the set of drawn strokes. Alvarado's system does suffer from somewhat lower accuracy values, probably due to the difficulty in training a DBN. Complex DBNs, as would be formed when encountering a large, complex sketch, contain many free parameters, resulting in difficulty making classifications. Alvarado's work does not allow the user to specify customizable thresholds for how strict they wish constraints to be applied. Additionally, it's not clear that the way DBNs handles confidence values maps to user perception.

3. Methodology

In the LADDER language [Ham07, HD05], constraints are applied to a set of primitive shapes (i.e., lines, arcs, and curves) that are recognized by other sketch recognition systems [PH08], as in Figure 1. LADDER's constraints are computed on a binary basis (Equation 8). Each constraint is assigned a hard-coded threshold, and this threshold is used to determine if a constraint holds or not. Recognition uses a tree-based method for pruning the search space based on which constraints hold over the set of strokes and primitive shapes that have been drawn. Multiple interpretations are computed, but they are differentiated only by context, not by recognition confidence. In our confidence-based implementation of the constraints, we use the same thresholds implemented in LADDER. Instead of using the thresholds to make a binary decision, however, we use them to assign a confidence that measures how certain the system is that the constraint holds.

With a confidence value assigned to constraints and the recognized shapes built using those constraints, we are able to add additional functionality to a sketch recognition system that might have been more difficult, or impossible, in a system using only binary constraints. We can provide an ordered *n*-best list of recognition interpretations, with each interpretation having its own measure of confidence.

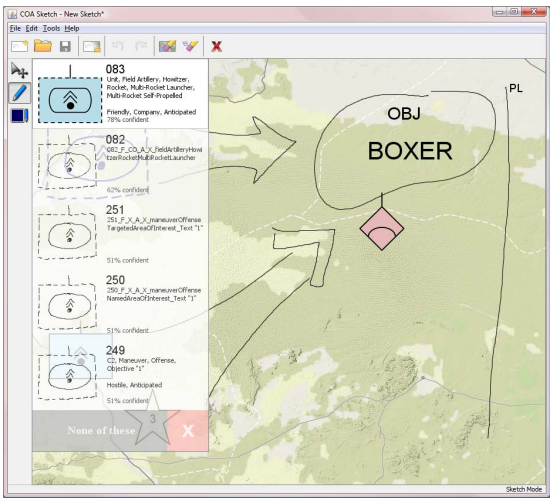


Figure 3: *An n-best list is presented for each symbol the user sketches in the interface shown in Figure 2. The confidences are computed, in part, using the confidences of the constraints that are used to compose each shape. By using an n-best list, we allow the user to pick the correct interpretation, even if the system makes a mistake and the correct interpretation isn't the one with highest confidence.*

Assigning confidence values to individual constraints also allows for more flexibility within the recognition algorithm itself. Rather than relying on binary decisions, an algorithm can use our constraints to allow for arbitrary levels of uncertainty in its decision and explore a larger solution-space. This might enable the algorithm to avoid local optima and find a globally optimal solution.

For example, consider the recognition capabilities illustrated in Figure 2. When recognizing the different parts of the COA sketch, we use the confidence values returned by the constraints to compute a list of different interpretations for the sketch. We then present the user with an ordered list of possibilities, enabling them to choose the correct interpretation out of a list in case the first interpretation was not correct. Figure 3 shows an example of the *n*-best list presented to the user, with the different confidences of interpretations for the given symbol. The confidence values allow us to order the interpretations, which is important because in a system with hundreds of shapes, there might be a large number of possible interpretations. Future plans for the system are to better integrate context with our confidence values to better automatically select the correct interpretation.

3.1. Distance-from-zero Constraints

We divide constraints into two categories. The first category uses a distance-from-zero approach to solving confidence.

For these types of constraints, the optimal maximal confidence is given when v, the input value to the constraint, is 0 (Equation 4). The confidence, c, for the constraint is computed using a half-Gaussian probability distribution (Figure 4(a)). A half-Gaussian distribution is simply a Gaussian distribution with mean $\mu = 0$, parameterized on the standard deviation σ^2, with input v constrained to be ≥ 0.

$$c = f_{\text{hg}}(v) \tag{1}$$

$$= \begin{cases} \frac{2}{\sigma\sqrt{2\pi}} \exp\left(\frac{-v^2}{2\sigma^2}\right) & v \geq 0 \\ 0 & v < 0 \end{cases} \tag{2}$$

$$= 2 * N(0, \sigma, v). \tag{3}$$

$N(\mu, \sigma, x)$ is the Gaussian/Normal probability distribution with mean μ and standard deviation σ, evaluated on the input x. For our constraints, we assign $\sigma = 1$. See below (Section 3.3) for a more detailed explanation of our choice of σ.

The constraint `horizontal` is an example of this type of distance-from-zero constraint. The value v for this constraint is relative to the absolute value of the angle of a line in relation to the x-axis (we do not use slope since the slope of a vertical line is $+\infty$). The line is perfectly horizontal if its angle is $v = 0$, since

$$\arg\max_v f_{\text{hg}}(v) = 0. \tag{4}$$

As v increases (the angle moves more toward vertical), the confidence c that the line is horizontal decreases toward 0.

3.2. Relational Constraints

The second category of constraints uses a relational approach. These constraints use a sigmoid function, illustrated in Figure 4(b), to compute confidence values.

$$c = f_{\text{sig}}(v) \tag{5}$$

$$= \frac{1}{1 + e^{-v}} \tag{6}$$

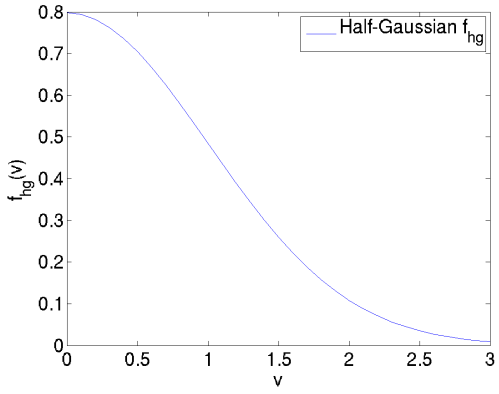
We call these constraints relational because they are used to compare two values, v_1 and v_2, against each other. We must consider the case of negative numbers, which we do not with f_{hg}. As v_1 grows larger than v_2, the confidence c increases toward 1. As v_1 grows smaller than v_2, the confidence decreases toward 0. The input to the constraint function is the difference in the values

$$v = v_1 - v_2. \tag{7}$$

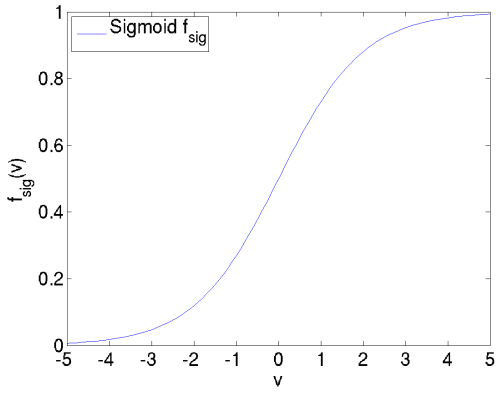
A value of $v = 0$ is ambiguous—it is not clear whether or not the constraint should hold since we cannot tell any difference between the values v_1 and v_2.

3.3. Normalizing v

We maintain the idea of a threshold as defined in the LADDER system's implementations of the various constraints. In LADDER, each constraint has a threshold, t, used to determine if the constraint was true or false.



(a) Half-Gaussian



(b) Sigmoid

Figure 4: *Two different confidence functions are used to compute the confidences for different types of constraints. Distance-from-zero constraints use a half-Gaussian confidence function (a), and relational constraints use a sigmoid (b). In both figures, the x-axis is the normalized value v' in Equation 10. The y-axis is the confidence value $c = f(v)$.*

$$c = f_{\text{LADDER}}(v) \tag{8}$$

$$= \begin{cases} 1, & v < t \\ 0, & v \geq t \end{cases} \tag{9}$$

We use these same thresholds in our system. However, instead of using them to make a binary decision, we use them for normalization purposes. There are many types of constraints, and many different threshold values of various magnitudes. In order to maintain the consistency of confidences returned by these different constraints, we normalize a value by the threshold for the constraint before solving for the con-

fidence. That is,

$$v' = \frac{v}{t} \tag{10}$$

and

$$c = f(v'). \tag{11}$$

If we did not perform this type of normalization, two constraints with different thresholds using the same confidence function f would not give the same confidence values, even if their inputs were comparable. For instance, `horizontal` has a threshold of 15°, while `acute-angle` has a threshold of 45°. Even though the two constraints both use the half-Gaussian confidence function f_{hg}, they will not return the same confidence value if both are exactly one threshold away from optimal. If we set $\sigma^2 = t$ for each constraint, then

$$f_{\text{hg}}(15) = 2 * N(0,15,15) = 0.0323$$

(for `horizontal`) and

$$f_{\text{hg}}(45) = 2 * N(0,45,45) = 0.011$$

(for `acute-angle`). But, if we set $\sigma^2 = 1$ and normalize our values as in Equation 10, then all constraints can be solved as

$$f_{\text{hg}}(1) = 2 * N(0,1,1) = 0.484$$

for values that are at the threshold. Similarly, we normalize the values for use in the sigmoid function f_{sig} to obtain consistent confidences. By normalizing in this way, we can picture constraint confidence as a measure of the "number of thresholds away from optimal" a given value is, ensuring a consistent space to compute confidence in regardless of the magnitude of the threshold.

3.4. Combining Constraints

When we determine that a high-level shape definition holds for a set of low-level primitives (its constraints are satisfied), we often want to set an overall confidence in our classification for that particular shape. That is, given a specific set of low-level shapes, how sure is the system that the high-level shape is present and all its constraints are satisfied?

We take a "weakest-link" approach to solving confidence for the entire shape, drawing from fuzzy logic theory, and let the high-level shape's confidence be the minimum confidence of the constituent constraints

$$c_{SHAPE} = \min_j c_j. \tag{12}$$

This method provides us with several appealing characteristics. First, it is independent of the number of constraints. This allows for arbitrarily complex constructions and considers simple and complicated sketches fairly with each other–we don't penalize the user in terms of recognition accuracy/confidence of our recognition for wanting to draw complicated things. At the same time, if a few constraints have a very low confidence, we aren't letting those low values get washed out in an average.

Other approaches are to assume the confidence values are probabilities over i.i.d. variables and multiply the n constraints together for a definition

$$c_{SHAPE} = \prod_{j=1}^{n} c_j. \tag{13}$$

This approach is reminiscent of Bayesian processes. It assumes that each shape occurs at random, and that a collection of a certain set of these random events denotes a high-level shape. The problem with this approach is that strokes are not made by a pen at random, and each stroke is influenced by the last when people are striving to create meaningful sketches, violating the independence assumption. Additionally, multiplying constraints always penalizes shape definitions with a larger number of constraints. While this makes sense when considering random variables, because a larger number of specific random events occurring together is less likely, it doesn't necessarily make sense for sketch recognition. The goal of sketch recognition is not to blindly analyze a series of strokes and see if they match a formulaic set of constraints. Rather, it is to enable the user to provide input to the computer through the pen. We don't want to make it harder on the user to enter arbitrarily complex drawings. We want to make it simpler. Thus, we cannot penalize more complex constructions through confidence multiplication.

We might also consider the average of all the constraint confidences and let that be the final shape's confidence

$$c_{SHAPE} = \frac{1}{n} \sum_{j=1}^{n} c_j. \tag{14}$$

However, this approach lacks the ability to significantly penalize a set of strokes that fails to meet one constraint out of a set of many, even if that one constraint is critical to the success of the shape definition.

3.5. Constraints Implemented

We have implemented 29 constraints from the LADDER shape description language in addition to 3 new constraints placed on the bounding box of a shape (for a total of 32), listed in Table 1. Five of these constraints use the relational approach for solving confidence, and all deal with the positioning of shapes in relation to one another. We use f_{sig} rather than f_{hg} for these constraints because there is not a good point at which to define a "zero" to use the distance-from-zero approach. For the rest of the constraints, there is a point at which you can define a baseline zero, and f_{hg} works well for computing confidence.

4. Results

In this section we discuss the results of our analysis of our methodology. We analyze the way in which we've imple-

Relational (f_{sig})	Distance-from-Zero (f_{hg})	
above	acute-angle	obtuse-meet
below	acute-meet	parallel
contains	bisects	perpendicular
left-of	closer	positive-slope
right-of	coincident	same-height
	connected	same-size
	equal-angle	same-width
	horizontal	same-x
	intersects	same-y
	larger-size	slanted
	negative-slope	smaller-size
	obtuse-angle	vertical
	bounding-box-tall	bounding-box-wide
	bounding-box-square	

Table 1: *The 32 constraints implemented with our confidence system. The five relational constraints deal with positions of shapes in relation to one another. The rest of the constraints use f_{hg} to solve confidence.*

mented the various constraints from two perspectives–user perception, where we see if our constraints match up with human intuition, and accuracy, where we make sure our system is useful for taking a set of low-level shapes and using the constraints between them to form a high-level target.

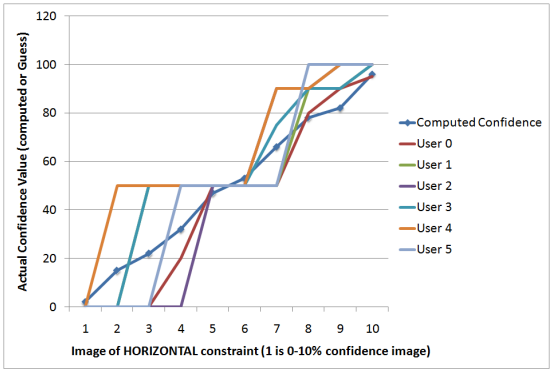
4.1. User Perception

The goal in sketch recognition is to assign classification labels to a set of strokes that agrees with human intuition. To this end we examined constraints placed on different low-level shapes drawn from a set of example sketches. The confidence values align themselves with our intuition.

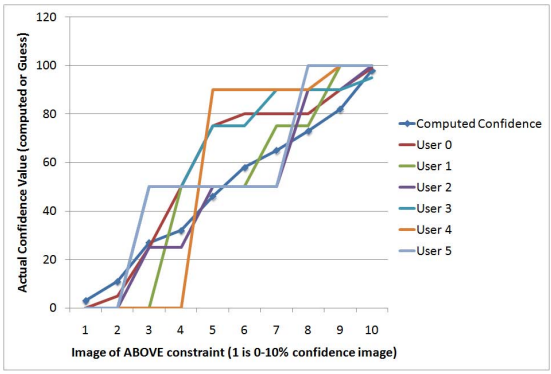
We showed six users ten examples each of images for the constraints `horizontal` and `above`, similar to those given in Figures 5 and 6. The user, unaware of the confidence value assigned to the shape, was asked how rate, on a scale of 0 to 100, how sure they were that "the line is horizontal" or "the red shape is above the blue shape." The shapes for each constraint were shown in a randomized order, to not bias the user (*i.e.* not in order of increasing confidence). Figure 7 shows the results from the study, and that user perception matches the general confidence values generated by these two constraints (the line denoted "Computed Confidence"). One note of interest is that people are not very gradated, and tended to use quartiles (*e.g.* 25%, 50%) rather than numbers in between. Also of interest was the fact that the humans produced monotonically increasing confidences as the angle increased, giving credence to the human-provided confidences.

4.2. Accuracy

Our constraint system has primarily been tested on sketches from the domain of military course of action (COA) symbols. We developed a recognition system that uses our constraints as part of the recognition process. We trained the



(a) Horizontal



(b) Above

Figure 7: *Plots for the responses to our user studies, one for the constraint `horizontal` (a) and another for `above` (b). The x-axes represent the image that was shown to the users, arranged in order of increasing confidence. There is one "Computed Confidence" line per plot, which represents the confidence assigned to that image by our system. The other lines are the confidences assigned by the participants in the user study.*

system by using a training set of over 6,000 hand-drawn examples of 485 different symbols. We modified the thresholds of the different constraints in the different shape definitions to maximize recognition accuracy. We then tested our system on a set of testing data consisting of 5,900 different examples. In total, our system achieved an accuracy of 89.9% accuracy overall. Note that this accuracy also includes any errors introduced by the low-level recognizers.

We found many cases where recognition using a constraint satisfaction approach became overly difficult. One specific COA symbol, for psychological operations, contains many constituent lines that fit together to form a "speaker" symbol (see Figure 8). The problem is that the recognition algorithm is trying to figure out which lines in the sketch belong to a speaker. So, it must perform a search over the

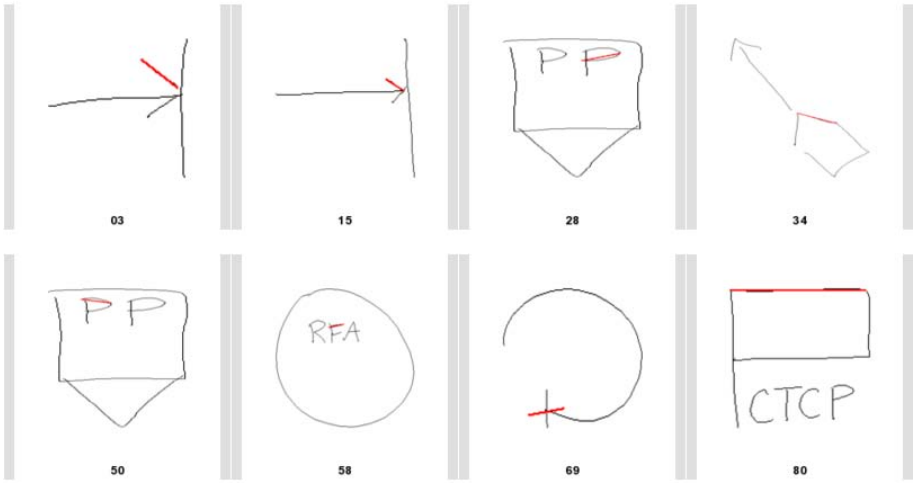


Figure 5: *Images of the* `horizontal` *constraint solved on various sketches from a military Course of Action symbol domain. Read the images as "Is the red line horizontal?" Our system's confidence for the answer "Yes" is given below each image.*

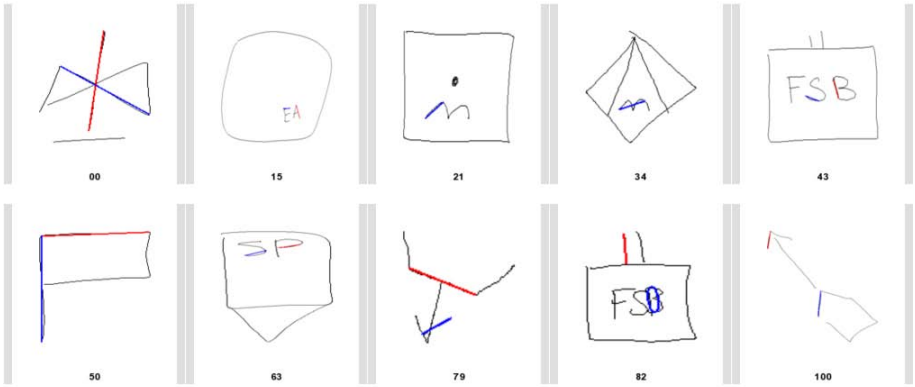


Figure 6: *Images of the* `above` *constraint solved on various sketches from a military Course of Action symbol domain. Read the images as "Is the red shape above the blue shape?" Our system's confidence for the answer "Yes" is given below each image.*

exponential problem space (the underlying problem is NP-complete) to see if there is a set of lines that satisfy the given constraints. For symbols like these, it quickly becomes infeasible to use a hierarchical, bottom-up approach for recognition, due to the sheer size of the search space.

There are also difficulties that arise when trying to tweak the customizable thresholds in the definitions for the shapes. It is often the case that in the initial version of a shape definition (as in Figure 1(a)), constraints are formulated by the programmer using an iconic picture of the symbol. In practice, these initial constraints are much too tight to be applied to an actual sketch. Humans are imprecise, and sketches made on a digitizing surface are inherently noisy. Using a training set, however, the programmer can fine-tune the thresholds to maximize accuracy. For one symbol in particular, wheeled armor, there are three circles that should fall `below` a larger ellipse. If the circles are defined to be completely below the ellipse, the recognizer will be too stringent and not recognize perceptually valid sketches, as illustrated in Figure 9. Loosening the threshold on `below` for this shape, however, can result in false positives. Instead, one has to constrain not the entire circle itself, but perhaps the centers of the circles. There are many tricky nuances when trying to hand-define constraints for shape definitions.

5. Future Work

For future work, we would like to perform user studies to verify that our confidence values are assigned in a manner

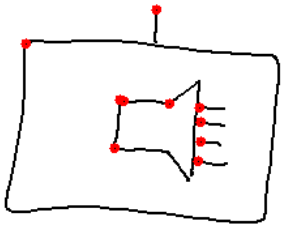
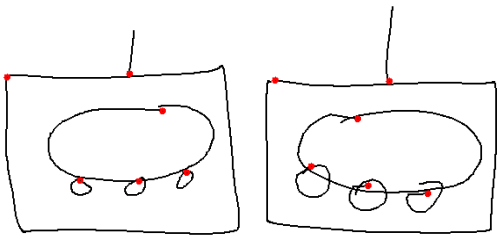


Figure 8: *Sketch of a psychological operations COA symbol. The nine lines in the center of the sketch that form the "speaker" result in a state-space combinatoric explosion, making the use of hierarchical, bottom-up recognizers problematic in terms of running time. Single instances of these sketches could take upwards of sixty seconds of recognition time.*



(a) Circles `below` (b) Circles not `below`

Figure 9: *It is apparent that, perceptually, these two sketches are both for the same COA symbol, wheeled armor. However, if one constrains the wheels to be* `below` *the ellipse, false negatives can occur ((a) would pass, while (b) would probably fail). Care must be taken then, to define the constraints properly. In this case, perhaps only constraining the centers of the wheels to be* `below`*.*

that aligns itself with human intuition and interpretation. We would also like to try learning appropriate confidence distributions from verification studies, as opposed to assuming a half-Gaussian or sigmoid function. Machine learning techniques could be used to learn appropriate thresholds from labeled data rather than relying on the empirical thresholds defined in LADDER.

6. Conclusion

We have described our method for computing confidence values for two kinds of constraints. Our method allows for threshold customization, which gives the users of sketch recognition languages and grammars the ability to customize how loose or stringent they want sketches to be drawn before they are recognized. Our method also allows for arbitrary levels of uncertainty within a recognition algorithm, as one can use the confidence values to prune different branches of the search space. Our constraints match user perception and have been used to build large-scale sketch recognition systems.

Acknowledgements

This work is supported in part by NSF grants 0757557 and 0935219. The authors would like to thank members of the Sketch Recognition Lab for their help in the completion of this work.

References

[AD04] ALVARADO C., DAVIS R.: SketchREAD: A multi-domain sketch recognition engine. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)* (New York, NY, USA, 2004), ACM, pp. 23–32. 2, 3

[Ham07] HAMMOND T.: *LADDER: A Perceptually-Based Language to Simplify Sketch Recognition User Interfaces Development*. PhD thesis, Massachusetts Institute of Technology, 2007. 3

[HCD*10] HAMMOND T., COREY P., DIXON D., EOFF B., JOHNSTON J., LOGSDON D., PAULSON B., PESCHEL J., TAEL P., WOLIN A.: A sketch recognition system for recognizing free-hand course of action diagrams. In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence (IAAI) (to appear)* (2010). 2

[HD05] HAMMOND T., DAVIS R.: LADDER, a sketching language for user interface developers. *Computers & Graphics 29*, 4 (2005), 518–532. 1, 3

[KK06] KIM D. H., KIM M.-J.: A curvature estimation for pen input segmentation in sketch-based modeling. *Computer-Aided Design 38*, 3 (2006), 238 – 248. 1

[LKS07] LEE W., KARA L. B., STAHOVICH T. F.: An efficient graph-based recognizer for hand-drawn symbols. *Computers & Graphics 31*, 4 (August 2007), 554–567. 3

[LM95] LANDAY J., MYERS B.: Interactive sketching for the early stages of user interface design. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)* (New York, NY, USA, 1995), ACM Press/Addison-Wesley Publishing Co., pp. 43–50. 2

[PH08] PAULSON B., HAMMOND T.: PaleoSketch: Accurate primitive sketch recognition and beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI)* (New York, NY, USA, 2008), ACM, pp. 1–10. 1, 3

[SPRN02] SHILMAN M., PASULA H., RUSSELL S., NEWTON R.: Statistical visual language models for ink parsing. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI)Spring Symposium on Sketch Understanding* (2002), AAAI Press, pp. 126–132. 2, 3

[SSD01] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: early processing for sketch understanding. In *Proceedings of the Workshop on Perceptive User Interfaces (PUI)* (New York, NY, USA, 2001), ACM, pp. 1–8. 1

[WEH08] WOLIN A., EOFF B., HAMMOND T.: Shortstraw: A simple and effective corner finder for polylines. In *Proceedings of the Eurographics Workshop on Sketch based interfaces and modeling (SBIM)* (2008), pp. 33–40. 1