Recognising Sketches of Euler Diagrams Augmented with Graphs

Gem Stapleton Aidan Delaney Peter Rodgers Beryl Plimmer
University of Brighton University of Kent University of Auckland
{g.e.stapleton,a.j.delaney}@brighton.ac.uk p.j.rodgers@kent.ac.uk bpli001@ec.auckland.ac.nz

Abstract

Euler diagrams form the basis of many visual languages. Such languages are formed by augmenting Euler diagrams with graphs or shading. However, tool support for creating augmented Euler diagrams is generally limited to generic diagram editing software using mouse and keyboard interaction. A more natural and convenient mode of entry is via a sketching interface which facilitates greater cognitive focus on the task of diagram creation. Previous work has developed sketching interfaces for Euler diagrams. This paper presents the first sketch tool for Euler diagrams augmented with graphs and shading. The tool allows the creation of sketches of these diagrams via pen-based interaction. To effect the recognition process, we define heuristics for classifying strokes as either curve, node, edge, or shading. To evaluate our recognition engine, we asked 10 participants to each sketch six augmented Euler diagrams. Using the results of the study, we fine-tuned the recognizer, achieving a statistically significant improvement.

1. Introduction

The effective use of visual languages often relies on software support. For example, in software engineering, class diagrams can be automatically generated from source code, allowing the programmer to visualize the structure of their implementation. From a different perspective, other software that supports the use of visual languages facilitates the creation of diagrams by users. For instance, standard diagram editors, such as Visio, allow users to draw diagrams using traditional point, click and drag mouse interaction.

Recently, we have seen the development of software that allows users to create diagrams using a stylus, or pen, on a touchscreen device. Sketching diagrams is advantageous in that it allows the user to focus on the actual diagram creation rather than the interface of the editing tools, and it is a useful problem solving and communications technique [4]. Sketch recognition software developed to date has focused on user interface design and graph oriented diagrams [7]. With respect to user interface design tools, the sketched items are largely independent of each other. In graph oriented diagrams, the spatial positioning of nodes and edges is not of semantic significance.

We have devised sketch recognition software for Euler diagrams [2, 15] where the spatial relationships between sketched items is fundamental to their semantics. This lays the foundations for the development of sketch recognition software for visual languages that extend them such as the Euler diagram augmented with a graph in figure 1; this is a variation on a diagram seen in [11] and represents information from a semantic network. Example notations that extend Euler diagrams include the monadic family of diagrammatic logics such as Swoboda and Allwein's Euler/Venn system [14], Shin's Venn-II system [13], spider diagrams by Howse et al. [5] and Gil et al. [6] and Minoshima et al.'s extended Euler diagram logic [9]. These diagrams typically augment Euler diagrams with graphs or shading, usually both. Furthermore, diagrammatic logics with greater expressive power add even more syntax, such as arrows, to make semantically rich statements. Examples include Kent's constraint diagrams [8] and Oliver et al.'s ontology diagrams [10].

We report on the development of sketch recognition software for Euler diagrams augmented with graphs and shading, which provides a basis for more sophisticated sketching software for the diagram types just given. Section 2 overviews our sketching software. The recognition component of our tool is detailed in section 3. Section 4 presents a user study used to evaluate and improve the recogniser. The software can be downloaded from [12].

2. SpiderSketch

The tool, called SpiderSketch, that we have developed allows users to sketch Euler diagrams augmented with graphs and shading and then recognises each pen stroke. Formally, *augmented Euler diagrams* comprise a finite set of closed curves drawn in the plane, a finite set of nodes, a finite set of edges whose end points are incident with nodes, and a set of minimal regions (formed by the curves) that are shaded. A

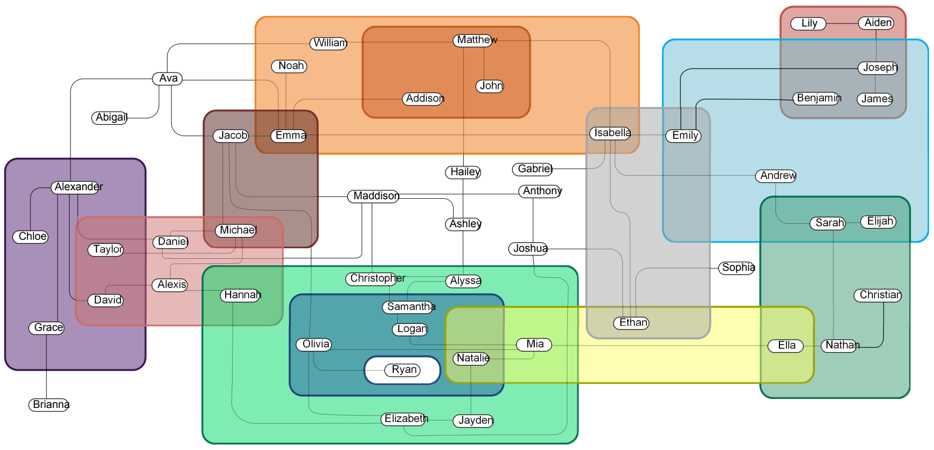

Figure 1. Visualizing semantic networks.

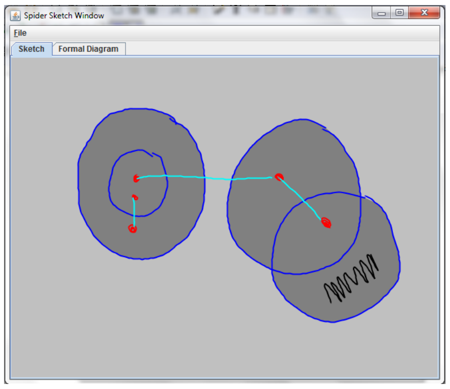

Figure 2. A screen-shot of SpiderSketch.

sketched diagram is a hand-drawn image that approximates an augmented Euler diagram.

In the sketching interface, users create sketches by drawing as on a piece of paper; figure 2 shows a screenshot. A stylus stroke is immediately rendered on the canvas. On completion of the stroke (stylus up event) it is passed to a recognizer that we describe in section 3. The recognizer result may be one of four classes: curve, node, edge, or shading. Each piece of syntax is immediately coloured to show the result of the recognizer. Curves are dark blue, nodes are red, edges are light blue, and shading is black. Online colouring allows one to readily check the association made by the software and, thus, have an opportunity to change the sketch if necessary. The sketch can also be saved.

3. The Recognizer

We have devised a number of heuristics that classify each stroke as one of the four types of syntax. The recogniser works online, classifying each stroke as it is created by the user. To design the heuristics, we identified features of each syntax type that distinguish it from the other syntax types.

1. Nodes are generally small and round, unlike the other pieces of syntax.

2. Shading is entered into the interface by a ‘scribble’ type action. This can be described as a stroke that is relatively long compared to the size of its bounding box and has many sharp turns. Potentially, this is similar to a node but we assume that shading takes up more space. Edges do not generally have sharp turns. Finally, curves are not particularly long compared to the size of their bounding box.

3. Edges are line segments that do not self-intersect. In addition, the end points are often far apart from one another, given the length of the line segment. By contrast, end points are generally close to each other for nodes or shading, which are ‘densely packed’ in to a space on the screen. In addition, curves should self-intersect since they are closed although users may not enter them that carefully; either way, the end points of a curve should be relatively close together.

The heuristics are described in full below and they are applied in a particular order. First, the recogniser decides whether the entered stroke, s, is a node. If s is not a node then the recongiser determines whether it is shading. Again, if s is not shading then it determines whether it is an edge. Finally, if s is not an edge then it is deemed to be a curve. Our heuristics are each reliant on a threshold value particular to it. For instance, we consider a stroke to be very small, and therefore a node, if the longest side of its bounding box is at most $T_{n,vs}$ pixels (n for node, vs for very small). Initially, we chose $T_{n,vs}$ to be 10 pixels. The threshold values can be adjusted to improve the recognition success rate. In full, the heuristics are as follows:

1. Any stroke that either (a) has a very small bounding box, or (b) is approximately square and has small bounding box is a node. By very small, we mean

$$BBlongestSide < T_{n,vs} \tag{1}$$

for some threshold value $T_{n,vs}$, where $BBlongestSide$ is the length of the longest side of the bounding box. By approximately square, we mean

$$\frac{BBshortestSide}{BBlongestSide} > T_{n,sq} \tag{2}$$

for some threshold value $T_{n,sq}$, where $BBshortestSide$ is the length of the shortest side of the bounding box. By small we mean

$$BBlongestSide < T_{n,s} \tag{3}$$

So, for a stroke, s, to be a node either (1) is true or both (2) and (3) are true.

2. Any stroke whose density is above a certain threshold, $T_{s,d}$, and contains at least $T_{s,t}$ sharp turns, typically called corners, is shading. We measure density by comparing the length of the stroke with the perimeter of its bounding box: the stroke must be significantly longer than the perimeter. In particular, we must have

$$\frac{\text{stroke length}}{\text{bounding box perimeter}} > T_{s,d}. \tag{4}$$

A corner is a single point, p, in the sketched line, l, where the angle, $\theta(p)$, formed by the two line segments on l whose end points are p satisfies

$$\theta(p) > T_{s,a} \tag{5}$$

for some threshold angle $T_{s,a}$. We count the number of corners and compare the number of them to a threshold value, $T_{s,t}$:

$$|\{p \in setP(l) : \theta(p) > T_{s,a}\}| > T_{s,t} \tag{6}$$

where $setP(l)$ is the set of non-end points on l. So, for a stroke, s, to be shading it must not be a node and both (4) and (6) must be true.

3. Any stroke whose end-points are far apart, given its length, and does not self-intersect is an edge. By far apart, we mean

$$\frac{\text{distance between end points}}{\text{length}} > T_e. \tag{7}$$

So, for a stroke, s, to be an edge it must not be a node or shading and (7) must be true.

Threshold	Value
$T_{n,vs}$	10 pixels
$T_{n,sq}$	0.55
$T_{n,s}$	15 pixels
$T_{s,d}$	0.5
$T_{s,a}$	60 degrees
$T_{s,t}$	1
T_e	0.2

Table 1. Initial threshold values.

Threshold	Value
$T_{n,vs}$	15pixels
$T_{n,sq}$	0.55
$T_{n,s}$	18 pixels
$T_{s,d}$	0.5
$T_{s,a}$	90 degrees
$T_{s,t}$	1
T_e	0.2

Table 2. Final threshold values.

4. Any other stroke is a curve.

The initial values we picked for these thresholds are in table 1. In section 4 we describe a study used to evaluate our recognition engine using these initial values. The results of the study were used to fine-tune the threshold values and the final values are shown in table 2. The process by which these final values were obtained is also described in section 4. To provide some insight into the recognition process, figures 3, 4, and 5 show a series of strokes, entered using the final threshold values, that show how the classification changes as properties of the strokes are altered.

4. Evaluating the Recogniser

We have conducted a study to evaluate the effectiveness of the recogniser component of our sketching tool. At the beginning of the study, each participant was given a short tutorial in how to use SpiderSketch. The researcher gave them a demonstration of the tool after which they were asked to draw some sketches, ensuring they were happy with how to

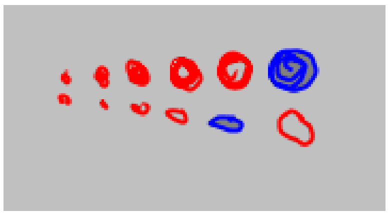

Figure 3. Recognising nodes.

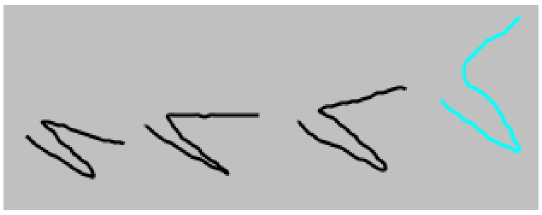

Figure 4. Recognising shading.

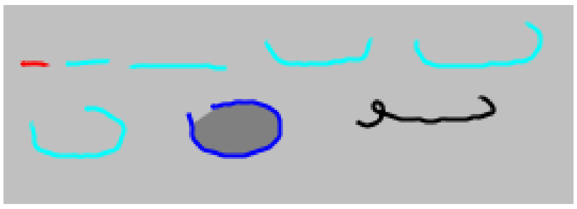

Figure 5. Recognising edges.

enter each type of syntax. In addition, they were told that it was a single stroke recogniser. In particular, they were informed that they had to remove the pen from the screen before drawing the next item. They were also told that if syntax was miscoloured then this was not their fault and they should not attempt to correct the stroke, because the study was about determining the correctness of the recognition engine. Each participant began the study when they were happy with using the tool.

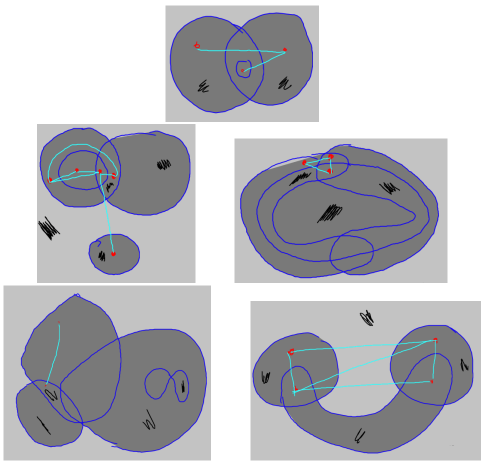

Figure 6. Diagrams used in the study.

For the study, each participant was asked to sketch six diagrams. The first five diagrams were given for them to copy and are shown in figure 6; each of these sketches was drawn by one of the 10 participants. For the sixth diagram, they were asked to draw any sketch with 3 curves, 3 nodes, 3 edges between the nodes, and 3 pieces of shading. An example diagram drawn by another participant can be seen

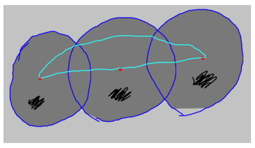

Figure 7. A sketch by a participant.

Syntax	Correct	Incorrect
Node	169	41
Not a node	587	4

Table 3. The node heuristics: initial classification.

in figure 7. Of note is that sometimes participants failed to copy the entire sketch and sometimes drew extra items in their sketches. Overall, there were roughly 200 strokes for each syntax type. Each sketch was saved for analysis purposes and was examined to determine whether it was correctly recognised. In particular, each stroke was assigned its correct syntax type by the researcher as well as the syntax type assigned by the recogniser.

Out of 801 sketched items, 747 were correctly classified, giving a success rate of 93.26% using the initial threshold values. Our task is now to determine whether the threshold values used in the heuristics can be adjusted to improve the recognition rate. Since the recogniser works hierarchically, we first examined the classification of nodes. Table 3 shows the number of strokes classified either as a node or not as a node using the initial values of the recogniser, broken down by whether that classification is correct. We can see that many of the $801 - 747 = 54$ original errors occurred in this part of the recognition process, with a total of $41 + 4 = 45$ errors. We extracted, from the saved sketches, all strokes incorrectly classified either as not being a node when it was a node (false negatives) or as being a node when it was not a node (false positives); these strokes can be seen in figures 8 and 9 respectively. In figure 8, some of the incorrectly recognised nodes are large and they break the condition that their bounding box is small. In figure 9, the strokes incorrectly recognised as nodes are generally small and it is perhaps unsurprising that errors occurred here. To improve the recogniser, we modified the values of $T_{n,vs}$ to

Syntax	Correct	Incorrect
Node	199	11
Not a node	576	15

Table 4. The node heuristics: final classification.

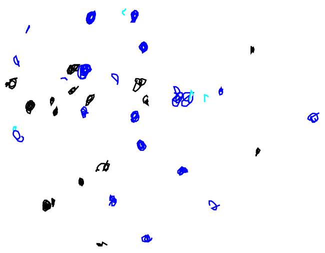
Figure 8. Strokes which should be nodes.

Figure 9. Strokes which should not be nodes.

15 and $T_{n,s}$ to 18, which were the best values we found after some exploratory data analysis. Altering the value of $T_{n,sq}$ did not seem to positively impact the recognition accuracy. Table 4 shows the numbers of correctly and incorrectly classifies strokes using these new threshold values for the nodes heuristics. There are now more false positives (increasing from 4 to 15) but many fewer false negatives (decreasing from 41 to 11).

Having changed the values of the thresholds associated with the node heuristics, we investigated classification errors associated with shading. At this point, we cannot overcome recognition errors due to the node heuristics. Thus, we now restrict our analysis to the only strokes that are correctly not recognised as nodes; this leaves $801 - 199 - 11 - 15 = 576$ strokes in the data set. At this point, there are no strokes which should have been recognised as shading, but were not recognised as such and only seven strokes that are shading but not recognised as such; see table 5 and figure 10 which shows the seven incorrectly classified strokes. Thus, we only attempt to improve the number of false negatives. Observing figure 10, we can see that some of these strokes do not have at least two sharp turns. Thus, we modified the shading threshold $T_{s,a}$ which is the angle used to determine whether a turn is sharp. The best value we found for this was 90 degrees. Table 5 shows the result of making

Syntax	Correct	Incorrect
Shading	176	7
Not shading	393	0

Table 5. The shading heuristics: initial classification.

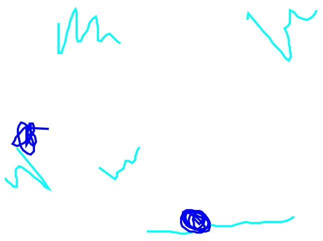
Figure 10. Strokes which should be shading.

Syntax	Correct	Incorrect
Shading	181	2
Not shading	393	0

Table 6. The shading heuristics: final classification.

this change, where no false positives were introduced and only 2 false negatives remain. These two incorrectly classified strokes can be seen in figure 11; since they are close to lines, it would be hard to distinguish them from edges.

Considering now the remaining data, i.e. the 393 strokes that are correctly identified as not being nodes or shading, the recogniser must determine whether each stroke is an edge. In fact, the recogniser has a 100% success rate here, correctly recognising 191 strokes as lines and the remaining 202 strokes are, by default, recognised as curves.

Using the final threshold values (table 2), of the 801 data points, 773 were correctly classified (totalling 11+15+2=28 errors), giving a final success rate of 96.5%. In order to determine whether the improvement is statistically significant, we carried out a McNemar χ^2 test (similar to a χ^2 test, but for paired data) using the data in table 7. Computing

$$\frac{(b-c)^2}{b+c} = \frac{(9-35)^2}{9+35} = 15.36$$

and comparing with a χ^2 table with one degree of freedom, we get a p-value of 0.0000886687. Therefore, the improvement is highly significant.

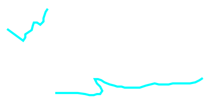
Figure 11. Strokes which should be shading after improving the recogniser.

		Final	
		Correct	Incorrect
Initial	Correct	$a = 738$	$b = 9$
	Incorrect	$c = 35$	$d = 19$

Table 7. Data for the McNemar test.

5. Conclusion

This paper presents the first tool, SpiderSketch, that supports users in the creation of augmented Euler diagrams. It uses a set of heuristics that determine whether each sketched item is a node, shading, an edge, or a curve. The recogniser was evaluated and improved using a sample of sketches produced by 10 participants. The improvements have been shown to be highly statistically significant. An accuracy rate of 96.5% is very high for a rules-based recogniser. Previous work [3] devised a recogniser, called CALI, for geometric shapes, possibly drawn using multiple strokes, such as lines, circles, rectangles and triangles. Whilst similar to our work here, it is our understanding that CALI would recognise each shape as some geometric shape, whereas for Euler diagrams it is important that some curves are recongised as just that, and not some particular shape.

Future work includes conducting another study, to reevaluate the recongiser against new data, with the sketches drawn by different participants. This will provide insight into how accurately the recogniser works after the changes we have made. We acknowledge that there is a danger of over-fitting the recogniser to the data set considered in this paper. In addition, we plan to improve the recognition by taking into account context. We want to explore a blend of online recognition with contextual re-recognition. In our case, even after the improvements we made, there were still misclassifications occurring between nodes and shading in particular. Since each edge should end next to a node, we can use this contextual information to reclassify items as more of the sketch is drawn. In addition, we also plan to extend the recogniser to more complex diagrams that include arrows and mathematical symbols. We expect context to play a bigger role in fixing classification errors in these more complex diagrammatic notations. Further, if we wanted to include text recognition then we would expect to use a more sophisticated recogniser such as RATA [1].

A further avenue of work is to extend the functionality of the tool so that is supports users given it's understanding of the sketch. In particular, we plan to include a feature that automatically produces a 'formal' version of the sketched diagram; by this, we mean a beautified sketch that looks as though it was drawn using an editing tool. Moreover, we want to incorporate syntax matching, so that any semantic differences introduced when converting sketch to formal can be automatically identified and rectified, as we have done for Euler diagrams in [15].

Acknowledgement This research is supported by EPSRC grant EP/H048480/1 and a Royal Society of New Zealand Marsden Grant.

References

[1] S. Chang, B. Plimmer, R. Blagojevic. Rata.ssr: Data mining for pertinent stroke recognizers. In *Sketch Based Interface Modeling*. ACM, 2010.

[2] A. Delaney, B. Plimmer, G. Stapleton, P. Rodgers. Recognising sketches of Euler diagrams drawn with ellipses. In *Visual Languages and Computing*, pp 305–310. Knowledge Systems Institute, 2010.

[3] M. Fonseca, C. Pitmentel, J. Jorge. CALI: An Online Scribble Recongiser for Calligraphic interfaces. AAAI Spring Symposium, 2002.

[4] G. Goldschmidt. *Visual and Spatial Reasoning in Design*, chapter The Backtalk of Self-Generated Sketches, pp 163–184. University of Sydney, 1999.

[5] J. Howse, G. Stapleton, and J. Taylor. Spider diagrams *LMS Journal of Computation and Mathematics*, 12(3):299–324, 2005.

[6] J. Gil, J. Howse, S. Kent. Formalizing spider diagrams *IEEE Symposium on Visual Languages*, pages 130-137, 1999.

[7] G. Johnson, M. Gross, J. Hong. *Computational Support for Sketching in Design*. Now Pub. Inc., 2009.

[8] S. Kent. Constraint diagrams: Visualizing invariants in object oriented modelling. In *Proceedings of OOPSLA97*, pp 327–341. ACM, 1997.

[9] K. Mineshima, M. Okada, Y. Sato, R. Taakemura, iagrammatic Reasoning System with Euler Circles: Theory and Experiment Design. In *Diagrams*, Springer, pp 188-205, 2008.

[10] I. Oliver, J. Howse, G. Stapleton, E. Nuutilal S. Torma, Visualising and Specifying Ontologies using Diagrammatic Logics. In *5th Australasian Ontologies Workshop*, CRPIT, pp 87-104, 2009.

[11] N. Riche, T. Dwyer Untangling Euler diagrams. IEEE Transactions on Visualization and Computer Graphics, 16(6):1090-1099, 2010.

[12] *www.cem.brighton.ac.uk/users/ges9/SketchingEuler Diagrams/SketchingEulerDiagrams.html* . 2011.

[13] S.-J. Shin. *The Logical Status of Diagrams*. 1994.

[14] N. Swoboda, G. Allwein. Heterogeneous reasoning with Euler/Venn diagrams containing named constants and FOL. *Euler Diagrams 2004*, ENTCS, 2005.

[15] M. Wang, B. Plimmer, P. Schmieder, G. Stapleton, P. Rodgers, A. Delaney. SketchSet: Creating Euler diagrams using pen or mouse Accepted for *VL/HCC*, 2011.