EUROGRAPHICS Symposium on Sketch-Based Interfaces andModeling (2009)
C. Grimm and J. J. LaViola Jr. (Editors)

Automatic Evaluation of Sketch Recognizers

Paul Schmieder, Beryl Plimmer, Rachel Blagojevic
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand

ABSTRACT

We present our toolkit to automatically evaluate recognition algorithms. There are few published comparative evaluations of sketch recognition algorithms and those that exist do not provide benchmarking or direct comparisons because standardised data and an evaluation platform is not available. By unifying data collection, labelling and evaluation in one tool, fair, flexible and comprehensive evaluations are possible. Currently we have 6 existing recognizers integrated into this tool. With our initial evaluations of these recognizers we have observed that the context from which training data is taken has an effect on recognition success rates. These results suggest that an evaluation platform such as this is a powerful adjunct for sketch recognition research.

Categories and Subject Descriptors (according to ACM CCS): I.7.5 [Document Capture]: Graphics recognition and interpretation

1. Introduction

Fair and comparative evaluations of sketch recognizers have been difficult and circumstantial because of the lack of a general evaluation framework. There are tools to collect and label data [WSA07, PWJH08] and others to interface to multiple recognizers [SNK07]. However, to our knowledge, no tool provides an interface to plug in recognition algorithms and generate performance information using labelled data.

A number of recognition techniques for hand-drawn sketches have been proposed. The efficacy is difficult to judge because there is no comparative benchmarking between techniques and algorithms. In part this is because the recognizers have been developed for different problems. Nevertheless, it is difficult to compare recognizers without consistently labelled training (for those that require training) and test data and an automatic test platform.

We extend our sketch framework DataManager [BPGW08, BSP09], with a flexible evaluation platform into which recognition algorithms can be plugged-in and automatically tested. By adding a module to evaluate recognition algorithms, we create a toolkit which brings sketched data collections and recognition algorithms together. The extensions are primarily the development of an evaluation interface to accommodate and test recognition algorithms. We have also integrated a multi-stroke labelling mechanism to accommodate a wider range of recognizers.

2. Overview

A framework uniting data collection, labelling, an interface to recognizers and a method to capture recognition results can form the foundation for comparative evaluations of recognizers. It enables an efficient evaluation of algorithms and the determination of the best available algorithm for new sketch domains. To evaluate new algorithms, the platform can be employed by simply integrating the new algorithm and testing it on data other recognizers have been tested on. Additionally, by providing the means to collect new data and label it, new domains can be tested on the implemented algorithms to determine the most appropriate one. Furthermore, previously unknown effects, such as the effect of different training datasets reported later in this paper, can be observed.

To automate the evaluation of recognition algorithms a flexible framework is needed. An impartial evaluation is difficult when the participating elements differ in their scope of operation as well as in the underlying input and output constraints. A flexible platform which maximizes the manipulation of evaluation parameters does not completely solve the problem of impartial evaluations but gets closer to the solution. Up to now, when recognizers have been compared, one of two approaches has been taken: either accept the inconsistencies as a source of error or adjust the algorithms to accept similar input and produce similar output.

Moreover, the framework acts as a repository for a feature library, recognition algorithms and ink datasets. Currently, in many publications promoting a new algorithm, the evaluation is constrained to tests involving data which is only used for the immediate assessment. This leads to a limitation of its significance. The acquisition of labelled data and of existing algorithms to lessen these limitations is problematic. Collecting and labelling data is time consuming. In the majority of cases the published description of an algorithm is not detailed enough to guarantee an exact copy; this renders any comparison meaningless. In this project some existing recognizers have been created from the published descriptions, others have been obtained by personal request to the authors.

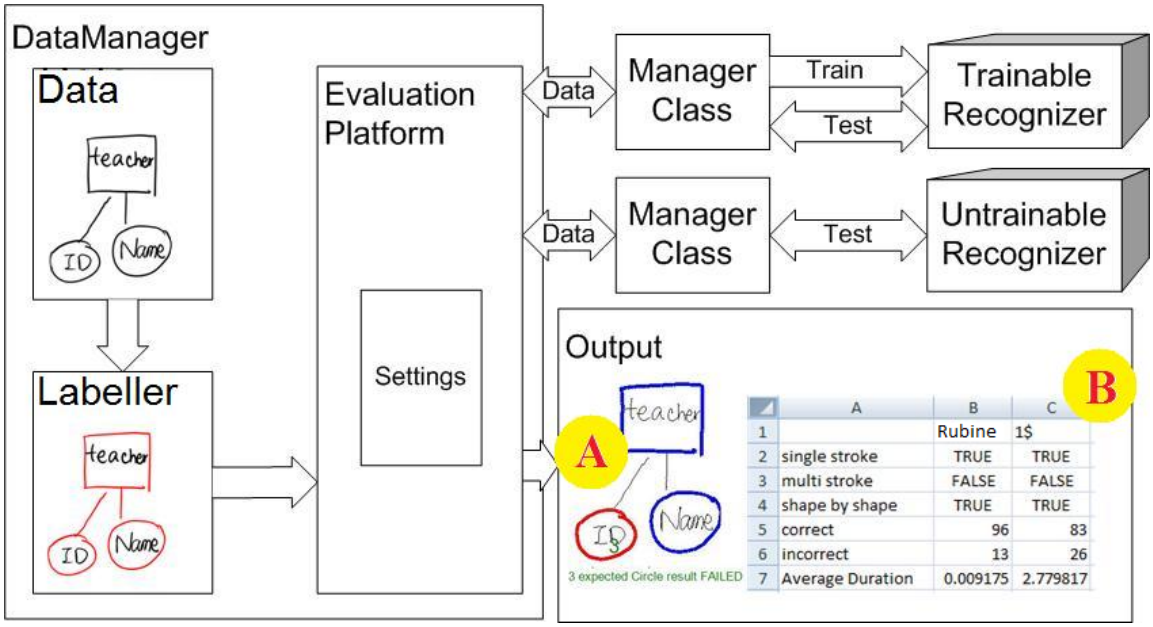


Figure 1: *The evaluation platform's architecture*

In the next section we present details of the evaluation platform implementation. We then report on two experiments we have used to evaluate the platform: for the first we integrated 6 basic shape recognizers and tested their performance over a limited set of basic shapes. The results suggested that the context of the drawings was affecting performance. The second experiment further explored this phenomenon. Finally we place this project in context with a review of related work and further discussion.

3. Implementation

The new evaluation platform is implemented into DataManager [BPGW08]. A number of extensions were needed to realize this platform; recognizer integration, training, testing, results generation and flexible labelling. The architecture of the new evaluation platform is shown in Figure 1.

The communication between the evaluation platform and the recognizers is handled by manager classes, each dedicated to one recognizer. The evaluation platform uses reflection to interact with the manager class that implements the necessary training and test methods to run the recognizer.

3.1 Recognizer Integration

As there is no standard programming language defined for recognition algorithms, DataManager (written in C#) has to be able to work with all possible languages; e.g. Microsoft C#, Java and C++.

To exchange information between the manager classes and the recognizers, data serialization is used. Data sent from the manager classes includes the component's strokes broken down into the point coordinates plus time stamps and the test settings. The serialization technique is also used for the manager classes to receive the recognition results from the recognizers.

To present data to different recognizer in appropriate formats it is then de-serialized and converted into appropriately formatted stroke objects. To run the recognition and serialize the results a script in the recognizer's programming language (Note: Only for non C# recognizers) has to be written by the user. This script can be called by the manager class once the necessary information has been serialized. To aid serialisation and de-serialisation in Java and C++ methods are provided as separate classes in the DataManager package.

Every recognizer must have a unique name so that it can be instantiated. The recognizer's name is stored in a settings file which is maintained by the user. Once a class is instantiated the methods to train and test the recognizer can be invoked.

3.2 Training

The effect of training a recognizer is to induce knowledge about the components it has to recognize. This is done by providing it with examples of the actual components. Depending on the algorithm's architecture, a recognizer may need to be trained. Ink data has to be provided to the recognition algorithm for this purpose. There are different learning techniques ranging from the creation of component snapshots [WWL07] to the generation of specially formatted files containing the serialized ink data [PF07]. As there are many different ways to train a recognizer the training method in the recognizer's manager class must be individually coded.

To select the data which is used to train the recognizer, DataManager iterates through all the sketches which have been selected for training. For each sketch, the components are checked individually as to whether they satisfy the evaluation options; e.g. single and/or multi-stroke components. The user specifies which participants to draw the training data from and the maximum number of training examples. In the case where there are more examples per component class available than allowed, a subset of the shapes has to be chosen by DataManager. Two different

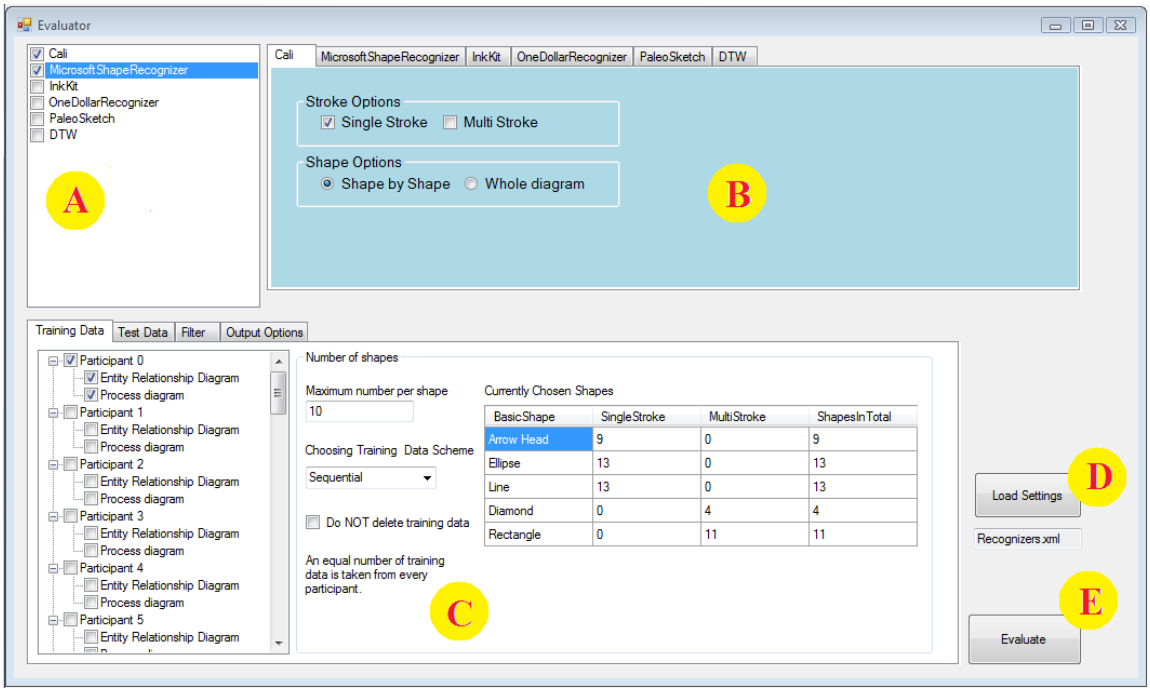


Figure 2: *The general interface of DataManager's evaluation platform*

methods have been implemented; random and sequential. Random selection can be useful when the influence of the selection of components within a sketch has to be examined or if the selection does not matter at all. However, the random method excludes the chance of repeating the exact same evaluation at a later point in time because there is no way to reselect exactly the same set of randomly chosen training examples. Sequential selection overcomes this problem by taking an equal number of shapes from every selected participant. With this approach diversity over the possible different drawing styles from different participants is guaranteed.

Components that pass the filter which eliminates prohibited components (components which have been excluded by the evaluation settings), trigger the recognizer's training method and are handed over to the recognizer together with training settings. The criteria a component has to satisfy for training are generally identical with those for testing. However if different criteria are required or training data in the corresponding format already exists, the location of this data can be specified in the settings file.

3.3 Testing

To test a recognizer, ink data and the test settings are input to the manager class's test method. The main task of the test method is to initialize and configure (according to the test settings) the recognizer, trigger the recognition and receive and store the results. Additionally, depending on the recognizer's implementation language, the manager may need to invoke the script which controls the recognizer in its language (see 3.1 Recognizer Integration).

The decision whether to pass the recognizer a complete sketch or the sketch's components one after another is based on the test settings. In contrast to the decision whether single and/or multi-stroke components are to be tested, this choice is mutually exclusive. If a complete sketch is passed to the recognizer it has to compute multiple recognition results and pass them back to DataManager. This is not necessary if one component at a time is processed.

To select the data which is used to test the recognizer, DataManager iterates through all the sketches which have been selected for testing, filtering and formatting the data using the same methods as are used for training data, and forwarding it to the recognizer. Once the recognition is finished, the results that consist of the recognizer's suggestions, the correct classification, duration and confidence values are stored.

3.4 Output

Once all recognizers are tested the results are prepared for output. Output can be generated in two different formats; screenshots of the wrongly recognized sketches and a Microsoft Excel file containing values and statistics from the evaluation.

A screenshot of a sketch is generated if any component has been incorrectly classified. To make the screenshot more readable each stroke is coloured depending on its recognition result. If a stroke has not been considered (because of test settings) it is coloured black. A correctly classified stroke is blue. Incorrectly classified strokes are coloured red and a number is assigned to every wrongly recognized component. A legend of these numbers below the screenshot gives the correct result and the algorithm's proposed result (see Figure 1 (A)).

The generated Microsoft Excel file contains information regarding several different test aspects. The information is grouped into 4 categories presented on four Excel worksheets; evaluation settings, general recognizer results, component results and participant analysis.

The first category, evaluation settings, provides information about the evaluation study so that the test can

be rerun under the same conditions. On the second worksheet, details about every recognizer's performance at different levels of granularity are given. At the most general level, overall performance information is given for all recognizers with the number of correctly and incorrectly classified components (Figure 1 (B)). To provide a more detailed analysis of the results a confusion matrix shows every recognizer's suggested classifications versus the correct result (see Figure 4 (B)). Statistics presented on the third and fourth worksheet give more detailed information on shape and participants respectively.

During the recognition process any program exceptions which are caused by the integrated recognizers are logged. The error.log file contains the information identifying the ink data causing the exception.

3.5 XML Settings File

The names of the integrated recognizers are provided to DataManager by including them in the XML settings file along with other recognizer specific settings. These settings are separated into two groups; options and mappings. An option specifies a setting related to either DataManager or the recognizer; for example whether the recognizer can accept multi-stroked shapes.

A mapping can be used to manipulate the recognition result statistics. With a mapping the name of the recognition result can be associated with the name of a test component. For example the component shown in Figure 3 may have been labelled in the test data as an "arrowhead" while Cali [FPJ02] calls it "move". To reflect this fact in the generated statistics, the mapping function is used.

Figure 3: *A basic shape called "move" by Cali and "arrow head" by DataManager*

3.6 GUI

From the evaluator GUI interface all elements necessary to set up and control an evaluation study of the recognizers can be set. Due to space constraints, only the most frequently used elements are always shown; the recognition algorithm list (Figure 2(A)), the button to load a setting file (Figure 2 (D)) and the button to start the recognition (Figure 2 (E)). The remaining elements are arranged on tabs. One set of tabs is for recognizer specific options (Figure 2 (B)) and the other one for test specific options (Figure 2(C)).

As every recognizer takes different forms of input, options to control these differences are provided. Via the options, those recognizers that can handle multi-stroke data can be tested on just single stroke examples. Also, some recognizers are restricted to one component per recognition step; others take complete sketches as input. These options are recognizer specific rather than test specific. By defining them as recognizer specific, it is possible to configure flexible tests such as the performance of recognizers restricted to single stroked components with those which can process multi-stroke components.

The lower tab set controls the data provided to the test for training and testing. Depending on the different aspects such as the number of training examples, the recognizer's performance may vary. The recognizer training options are: the selection of training data, the maximum number of training examples per component and the method the training examples are chosen.

The training data is selected by checking the participants and their sketches. It is not possible to isolate certain parts of the sketch; when a sketch is selected all its labelled components (dependent on the filter) become possible training examples. However it is possible to set a maximum number of training examples per component. This restriction is applied to every class of components but does not guarantee that enough examples of a component are available. To provide an overview of the component numbers included in the selected sketches, a table containing all the different types of labelled components is shown. The number of single stroke components is listed separately from the number of multi-stroke components (Figure 2 (C)).

The test tab is similar in structure and content to the training tab. While it is usual to use different data for training and testing this is not enforced, but a warning message is generated if the same data is selected.

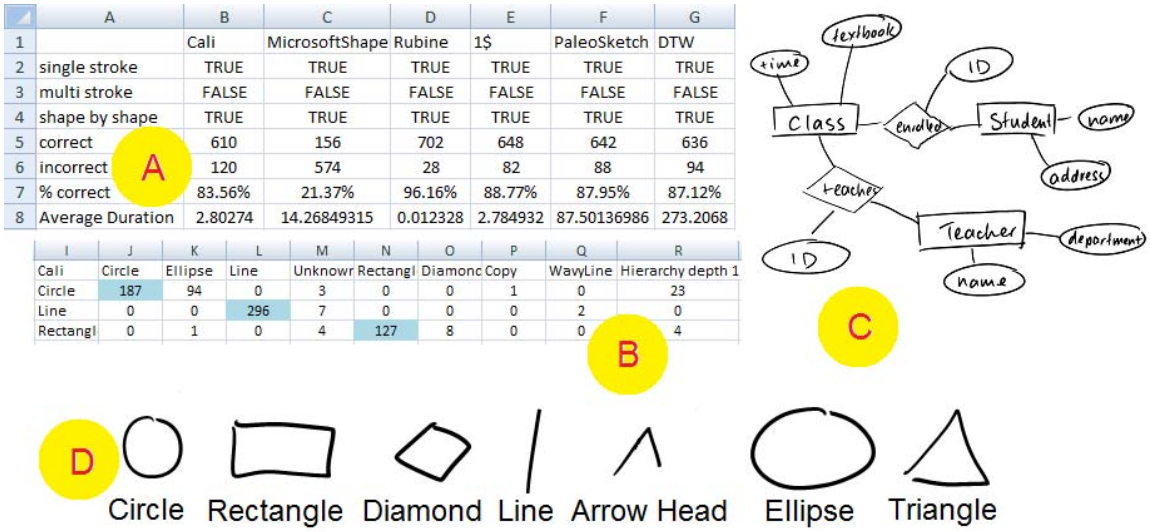
The next tab guides the filtering of basic components and is controlled by a check-list generated from the loaded dataset showing all the different labels which have been used to tag sketch components in the dataset. Component types to participate in an evaluation study are selected, the other classes are deselected. This filter applies for training as well as for testing.

3.7 Labelling

Besides the recognizer integration and test platform the DataManager labelling has been extended as some recognizers evaluate components consisting of more than one stroke (multi-stroke). The existing single-stroke labelling system has been supplemented with the new one which allows the grouping of multiple strokes to one component. Thus the same stroke can be part of different components; for example a rectangle could be labelled as rectangle, quadrilateral and square. Multi-stroke labels can be added, removed and manipulated at any time.

4. Experiment 1

As a proof of concept of the evaluation framework we have integrated and evaluated 6 basic shape recognition algorithms; Cali [FPJ02], Microsoft Ink Analyser, 1$ recognizer [WWL07], Rubine's [Rub91] with the extended feature set used in InkKit [PF07], PaleoSketch [PH08] and a recognizer using dynamic time warping techniques (DTW) [WWL07].



	A	B	C	D	E	F	G
1		Cali	MicrosoftShape	Rubine	1$	PaleoSketch	DTW
2	single stroke	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
3	multi stroke	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
4	shape by shape	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
5	correct	610	156	702	648	642	636
6	incorrect	120	574	28	82	88	94
7	% correct	83.56%	21.37%	96.16%	88.77%	87.95%	87.12%
8	Average Duration	2.80274	14.26849315	0.012328	2.784932	87.50136986	273.2068

A

	I	J	K	L	M	N	O	P	Q	R
1	Cali	Circle	Ellipse	Line	Unknowr	RectangI	Diamonc	Copy	WavyLine	Hierarchy depth 1
2	Circle	187	94	0	3	0	0	1	0	23
3	Line	0	0	296	7	0	0	0	2	0
4	RectangI	0	1	0	4	127	8	0	0	4

B

C

D

Figure 4: (A) *Evaluation summary results for line, rectangle and circle tests using 6 different basic shape recognizers. The duration is given in milliseconds.(B) Confusion matrix showing Cali's detailed classification results. (C) An ER diagram from DDS which's shapes can be used for trainingand testing. (D)All shape classes contained in both datasets; BSDS and DDS.*

This set of recognizers varies in implementation language (Java, C# and C++), recognition algorithms and techniques. For the Java recognizer (PaleoSketch) we interfaced to the jar file. C++ and C# recognizers have been referenced as libraries (i.e. Cali and the Microsoft Ink Analyser). The source code of the three remaining recognizers implemented in C# is directly integrated. Out of the six integrated recognizers, three have to be trained; Rubine, DTW and 1$. PaleoSketch has been provided to us on personal request, Cali has been downloaded from http://vimmi.inesc-id.pt/cali/code.html and DTW and 1$ from http://depts.washington.edu/aimgroup/proj/dollar/, for Rubine's we have used our group's InkKit implementation.

4.1 Basic Shapes

All of these recognizers seek to identify basic shapes found in diagrams. There is no definitive set of basic shapes. The most common are: line/curve/arc, rectangle/square/diamond, circle/ellipse, arrow head and triangle (Figure 4 (C,D)).

The larger the set the more difficult it is to accurately recognize basic shapes due to the increased number of possible candidate classes and similarities between classes. If a set only consists of two different shape classes there is a 50% chance that the correct result is computed by pure chance. The likelyhood of misclassifications increase even more in case of different shape classes that are similar such as ellipses and circles.

Another difficulty when recognizing basic shapes is a shape's composition. A basic shape can be sketched with an arbitrary number of strokes (multi-stroke) and furthermore, one stroke can represent an arbitrary number of shapes (complex shapes). However, complex shapes, as the name indicates, are not considered basic shapes.

4.2 Dataset

To conduct the experiment a data set of basic shapes has been used. The set contains six different drawings from 33 participants with examples of the basic shapes: circle, rectangle, diamond, arrow, triangle and ellipse. Each person drew 9 or 10 examples of each shape; no instruction was given to them on drawing style. We refer to this dataset as the Basic Shape Data Set (BSDS). For this experiment only single stroke lines, rectangles and circles have been used (see Table 1).

Table 1: *Overview of the basic shapes used for both experiments taken from the basic shape dataset (BSDS)*

	Single stroke	**Multi stroke**	**Total**
Circle	304	1	**305**
Rectangle	156	143	**299**
Line	324	1	**325**
Total	**784**	**145**	**929**

4.3 Experiment Settings

For the evaluation all recognizers have been tested on lines, rectangles and circles. Additionally, the basic shapes have been input one at a time, not as complete diagram sketches. Rubine's [Rub91], 1$ and DTW [WWL07] have been trained with 15 examples per shape class using two participants' sketches. The remainder of the data was used for testing. To run the experiment, a Dell Optiplex 775 running Windows Vista with an Intel Core Duo CPU with two 3.00 GHz cores and 4 GB Ram was used. The evaluation took 17 minutes and involved the test of 285 circles, 140 rectangles and 305 lines taken from table 1 column 2 (with the other data used for training). The results are generated as a Microsoft Office Excel file and colour coded images of the misclassified basic shapes.

The raw results can be seen in Figure 4 (A). Rubine's achieved the highest recognition rate closely followed by 1$, PaleoSketch, DTW and Cali. The Microsoft Ink Analyser performed considerably worse.

With these results we can demonstrate the difficulty of comparing recognizers. It is not always possible to manipulate a recognizer's basic shape set which makes a

fair comparison a difficult undertaking. Cali's basic shape set cannot be manipulated, in 94 cases circles have been classified as ellipses. In 23 of these 94 misclassifications the correct result has been returned as the second most likely result (Figure 4 (B), last column). In contrast, PaleoSketch has an option to switch off the modules responsible for a shape class. Figure 4 (A) shows the results for PaleoSketch when only the circle, rectangle and line modules are activated. However, in the case when all 14 modules are activated, PaleoSketch recognizes 638 shapes correctly, 44 less than before. Note that these numbers are not representative for any of the recognizers' real performances as we just use them to outline the power of DataManager and the difficulties of a fair evaluation. Other tests we have run with different combinations of shape sets have given quite different rankings. The evaluation also reports on the average classification time per shape in milliseconds given that the recognizer is already trained and configured. Large differences between the recognizers can be seen (Figure 4 (B) row 7).

The analysis from a shape's perspective shows that almost one third of all rectangles have been misclassified while 82% of the circles and lines have been correctly classified (Table 2).

Table 2: *Shape matrix for all recognizers*

	Circle	Line	Rectangle
Correct	1387	1502	561
Incorrect	323	328	279
% Correct	81.11%	82.08%	66.79%

5. Experiment 2

We also tried the same experiment on a diagram dataset (DDS) of Entity Relationship (ER) diagrams and Process diagrams drawn by the same participants and collected at the same time as the BSDS dataset used in experiment 1. This dataset consists of 33 ER and Process diagrams drawn by each participant from a text description of the requirements. We noticed quite different results for the non-trainable recognizers (Cali, PaleoSketch, Microsoft Ink Analyser).

We hypothesised that there may be some difference between participants' drawing style when drawing individual component examples to when they draw them as a part of a diagram.

To investigate our observations, we ran four more evaluations. For the first two tests trainable recognizers (Rubine, 1$ and DTW) have been trained on the basic shapes from participants 1 to 7 from the BSDS and all recognizers tested on participants' 8 to 33 data from both datasets. For the third and fourth test, the trainable recognizers have been trained again on participants 1 to 7's data from the DDS dataset and tested on participants' 8 to 33 data from both datasets. We modified the basic shape set from the first experiment replacing circles with ellipses as we had more examples of ellipses in the DDS.

When comparing the success percentages there are some clear break points in the difference in results. The recognition rates are either <2.5% different or greater than 5%. Our data collection process was not 'balanced' (we had not planned to do this experiment) so rather than using complex statistics we simply considered values less than 2.5 as worthy of further investigation. Tables 3- 5 show the summary results from this experiment.

Table 3: *Results for non trainable recognizers*

	Cali	MS	PaleoSketch
Test Diagrams (% Correct)			
Rectangle	91.30	65.22	93.48
Line	84.07	0.00	88.50
Ellipse	94.06	81.74	98.17
Test Basic Shapes (% Correct)			
Rectangle	90.43	22.61	78.26
Line	97.58	0.00	97.58
Ellipse	95.26	75.86	99.57
Differences			
Rectangle	0.87	42.61	15.22
Line	-13.51	0.00	-9.09
Ellipse	-1.19	5.87	-1.40

Table 4: *Results for trainable recognizers tested on diagram dataset*

	Rubine	1$	DTW
Train Diagrams (% Correct)			
Rectangle	82.61	91.30	82.61
Line	100.00	100.00	100.00
Ellipse	97.26	98.63	96.80
Train Basic Shapes (% Correct)			
Rectangle	67.39	71.74	80.43
Line	100.00	97.94	100.00
Ellipse	99.54	73.52	80.37
Differences			
Rectangle	15.22	19.57	2.17
Line	0.00	2.06	0.00
Ellipse	-2.28	25.11	16.44

Table 5: *Results for trainable recognizers tested on basic shape dataset*

	Rubine	1$	DTW
Train Diagrams (% Correct)			
Rectangle	92.17	87.83	76.52
Line	100.00	100.00	100.00
Ellipse	98.28	98.71	98.71
Train Basic Shapes (% Correct)			
Rectangle	77.39	65.22	71.30
Line	100	99.19	100.00
Ellipse	99.57	85.78	90.95
Differences			
Rectangle	14.78	22.61	5.22
Line	0.00	0.81	0.00
Ellipse	-1.29	12.93	7.76

Five of the nine results (differences) for the untrained recognizers (Table 3) are greater than 5%. For rectangles, Microsoft and PaleoSketch perform much better on DDS, while Cali is equally good. For ellipses, Microsoft achieves much better classification rates on DDS whereas PaleoSketch and Cali show similar results on both datasets. For lines, Cali and PaleoSketch perform better on the BSDC. This is worthy of further investigation.

For the trainable recognizers (Rubine, 1$ and DTW) we can compare results in two ways.

1. From each set which has been trained and tested on the *same* data; e.g. trained and tested on DDS (Table 4).

2. From each set which has been trained and tested on *different* data; e.g. trained on DDS and tested on BSDS (Table 4, Table 5).

While the first comparison does not show any obvious patterns, the second one where the performance on the datasets is compared with the different training sets shows very clear differences for closed shapes (Tables 4 & 5).

In most cases (the exception being Rubine's ellipse which was negative, but below the threshold) the recognizers performed considerably better when trained on diagram data. For the 1$ recognizer the difference was a stunning ~20%. Interestingly there was no difference in lines, such as we observed for the untrained recognizers, apparent with the trained recognizers. All values for lines were below the threshold.

6. Related Work

We know of no other tool to automatically test sketch recognition algorithms. There are tools for collecting and labelling data, generating feature sets and integrating multiple recognizers.

The closest match to this work is iGesture, a gesture recognition framework [SNK07] for integrating multiple recognizers. It works from the perspective of individual components (that may consist of several strokes). The framework provides interfacing to various recognizers, evaluation of individual gestures and an API for application programmers. The concept of standalone gestures is central to its architecture: gesture definition is the first step in describing a problem and the test bed is based on evaluation of standalone examples.

There are tools to collect data and tools to label it afterwards. SOUSA [PWJH08] is a tool which can be used to collect data. It is an online based application which allows researchers to create their collection studies defining the collection criteria. Once a project is set up, participants can use the tool to provide input in the form of sketches done according to the predefined task descriptions. The collected data can be accessed by everyone.

To label data a tool by Wolin et al. [WSA07] can be used. The tool provides grouping as well as fragmenting functions. Grouping is when multiple strokes are combined to one component. Fragmentation is the opposite process where a stroke is broken up into multiple components.

In DataManager [BPGW08 , BSP09] multiple labels can be applied to the same stroke. A hierarchical labelling approach is used which allows the inference of a predefined structure of sketch components thus saving time during the labelling process. This means that a label can be defined as a child of another label, thus inferring the affiliation of a component to multiple labels.

To our knowledge DataManager is the only tool which can be used to do both, data collection and labelling. DataManager also contains an automatic labelling mechanism dividing sketch components into shapes and writing [PPGI07]. Additionally, DataManager contains a library of features which can be used to generate feature sets from sketch datasets. The feature sets are not used in this project, but can be used in data mining tools such as Weka [WF02].

7. Discussion

In order to enable automatic evaluations of recognizers we have extended our DataManager with an evaluation platform. The integration is facilitated via a flexible software interface and settings file. This allows for recognizers written in different languages and data in different formats to be treated fairly.

iGesture [SNK07] has a similar approach to interfacing recognizers but a very limited evaluation interface. Their approach is gesture based, in comparision to our approach which is data driven. At the core of our architecture are complete diagram examples. The example diagrams are labelled and feed into the various algorithms for evaluation within the context of the diagram. There are two reasons to take this approach. First, we have found that inter-stroke data is useful in some problem contexts [PPGI07] and this is missing or unreliable when gestures are treated individually. Second, our evaluator can be used to evaluate algorithms that use rich contextual information.

As outlined in experiment 1, it is difficult to do a fair comparison of recognizers as each has been designed for different problems. We have noted with the basic shape recognizers integrated into DataManager some of the different constraints on input and different basic shape sets. These differences exist in other categories of recognizers. With basic shape recognizers it would not be fair to compare the results of test involving single and multi-stroke components as it is more difficult to recognize multi stroke components than single stroke ones. Furthermore, for both tests, single and multi-stroke, different data has to be used since a component either consists of one or multiple strokes. The same difficulty in judgement arises when component by component is passed to the recognizer in contrast to the whole sketch. A recognizer accepting an entire sketch could take advantage of the spatial ordering of the components but has to decompose the sketch in the first place.

Despite these difficulties, comparing the performance of different recognizers is a worthwhile activity. Recognizer comparative studies, like all experiments, must control all the variables except those of interest. With an automatic framework this can be achieved. The experiments that we have reported here took two datasets and explored the efficacy of the integrated recognizers within a limited range of basic shapes.

While these experiments are only indicative of the types of experiments that can be run, and used here as a proof of concept, they have produced some interesting results. It is clear from experiment 2 that the context of drawing is influential on the performance of algorithms. Most current tools that use trainable recognizers, take standalone examples for training. This is appropriate if we are interested in recognizing standalone examples. However if diagrams are the target, our results suggest that much higher success rates are possible with these recognizers if they are trained from example diagrams. One possible explanation could be that shapes in the BSDS have been sketched more tidily because users were focused on the task of drawing a shape rather than on drawing a diagram

contained of shapes. Therefore the feature measurements are less realistic.

Many other experiments could be simply configured with an automatic platform. For example numerous enhancements to Rubine's algorithm have been proposed e.g. [PF07, SNK07]. It would be relatively simple to implement each of these variations and compare their efficacy. A similar study with various stroke splitting and joining algorithms would also be possible.

Another reasonable aim of an evaluation can be to find the most suitable recognizer for a problem. If there is a new domain of sketches and the "best" integrated recognizer has to be determined, then the winner depends on the definition of best. If "best" means the recognizer with the highest accuracy, then highest accuracy wins. However, if the problem involves recognition of multi-stroke components and the manipulation of the recognizer's basic shape set, then the most accurate recognizer might not be the most suitable one.

Finding recognizers' strengths and weaknesses and then weighing every recognizer's performance against the new domain's demands can help to create a new recognizer. By analysing the parts of the tested recognizers, the pieces of code responsible for the good performance can be identified and extracted. Once all the "good" parts from the recognizers are gathered, they can be combined to make a better recognizer.

One limitation of the DataManager is that the degree to which a recognizer can be adjusted to a test varies. As shown in experiment one when using PaleoSketch [PH08] it is possible to manipulate the basic shape set by switching off the corresponding modules. The influence of this has been shown in the experiment and the extent to which it influences the accuracy. The same configuration is not possible with Cali and Microsoft Ink Analyser. However, such manipulation is possible for recognizers which have to be trained by only training on the components which have to be classified.

8. Conclusion

The recognition of sketches has been attempted by various research groups over the past 15 years. Yet a satisfactory recognizer in terms of accuracy and drawing constraints has not been found. To evaluate an algorithm, it is compared to others using the same data they used. Due to the lack of standardised datasets a meaningful comparison has not been possible. Furthermore, not all existing recognizers are publicly available, which restricts comparative evaluations in terms of significance. By extending the sketch framework DataManager [BPGW08] with an evaluation platform, we overcome the outlined problems by providing a tool with which recognition algorithms can be compared using the same datasets. The experiments we have conducted with the platform validate the recognizer integration strategies we have employed and demonstrate the types of data that can be produced.

Additionally, the integration of new recognition algorithms is supported and by doing so the tool will provide a steadily growing pool of integrated algorithms which can be used by others thus saving time and increasing the significance of the comparative evaluation studies.

The next step in the development of DataManager's evaluation platform will be to add various forms of training strategies such as k-fold cross validation. Furthermore, features to save test settings such as selected data for training and testing will be integrated.

9. Acknowledgements

Thanks to the people who provided their recognizers for this research. This research is partly funded by Microsoft Research Asia and Royal Society of New Zealand, Marsden Fund.

References

[BPGW08] BLAGOJEVIC R., PLIMMER B., GRUNDY J., WANG Y.: A Data Collection Tool for Sketched Diagrams, *5th Eurographics Conference on Sketch Based Interfaces and Modelling (SBIM '08)*, Annecy, France.

[BSP09] BLAGOJEVIC R., SCHMIEDER P., Plimmer B.: Towards a Toolkit for the Development and Evaluation of Sketch Recognition Techniques. *Intelligent User Interfaces (IUI '09),* Florida, USA.

[FPJ02] FONSECA M. J., PIMENTEL C. E., JORGE J. A.: CALI: An Online Scribble Recogniser for Calligraphic Interfaces. *AAAI Spring Symposium on Sketch Understanding*.

[PPGI07] PATEL R., PLIMMER B., GRUNDY J., IHAKA R.: Ink Features for Diagram Recognition. *4th Eurographics Workshop on Sketch-Based Interfaces and Modeling Riverside*, California: 131-138.

[PH08] PAULSON, B., HAMMOND T.: PaleoSketch: accurate primitive sketch recognition and beautification, *Proceedings of the 13th international conference on Intelligent user interfaces*. ACM, Gran Canaria, Spain.

[PWJH08] PAULSON B., WOLIN A., JOHNSTON J., HAMMOND T.: SOUSA: Sketch-based Online User Study Applet. *Sketch Based Interfaces and Modeling*, Annecy, France: 81-88.

[PF07] PLIMMER B., FREEMAN I.: A Toolkit Approach to Sketched Diagram Recognition. *HCI*, Lancaster, UK, 1: 205-213.

[Rub91] RUBINE D. H.: Specifying gestures by example. *Proceedings of Siggraph* '91: 329-337.

[SNK07] SIGNER B., NORRIE M. C., KURMANN U.: iGesture: A Java Framework for the Development and Deployment of Stroke-Based Online Gesture Recognition Algorithms.

[WF02] WITTEN I. H., FRANK E.: Data mining: practical machine learning tools and techniques with Java implementations. *ACM SIGMOD Record* 31: 76-77.

[WWL07] WOBBROCK J. O., WILSON A. D., LI Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes, *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, Newport, Rhode Island, USA.

[WSA07] WOLIN A., SMITH D., ALVARADO C.: A Pen-based Tool for Efficient Labelling of 2D Sketches. *4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, Riverside, CA.