Online Flowchart Recognition and Interaction

ASWIN JACOB THOMAS* and AMALA BABU*, CSCE624 Sketch Recognition, Texas AM University

Flowcharts are a sequence of symbols which represent information and logic. It is an easily understandable mode of representation for algorithms. In this project we have created a simple flowchart recognition system using ellipse, square, line, diamond and rectangle symbols to represent the different operations. The user drawn symbols are first recognized by the system and later replaced by a beautified version which can then be easily converted into a pseudo-code by creating a connected graph out of the drawing. The pseudo-code can then be extracted by traversing the graph. Code of the project can be accessed at https://github.com/aswin-jacob-thomas/FlowChart-Recognition. The system is deployed in AWS EC2 instance and can be accessed using http://flowchart.us-west-2.elasticbeanstalk.com/

CCS Concepts: • **Sketch Recognition**; • **Flowchart**; • **Rubine's features**; • **Paleosketch**; • **Pseudo-code**;

Additional Key Words and Phrases: accuracy, f-measure, paper.js

1 INTRODUCTION

Sketching is a process that invokes the creativity of a person. We can arrive at solutions to even complex problems by sketching its logic. Humans have always used images and sketches to communicate ideas, represent and describe data . Good examples are the music notations, free body diagrams in mechanics, shapes used in flowcharts etc. Using figures and drawing etches a concept deep in a person's mind compared to learning using only text.

Flowchart is a type of sketch that represents a workflow or process. It is a diagrammatic representation of a sequence of steps which when followed, arrive at a solution to a problem. It contains a set of pre-defined shapes, which correspond to a particular action, connected by arrows to represent the flow within it. Flowcharts are used by computer scientists to visually describe algorithms and for ease in understanding the logic behind it. It is used to teach the basics of programming to students.

Drawing a flowchart on paper is a tedious task because if we make a mistake, the whole diagram needs to be redrawn. Validation and correction of flowcharts is also time consuming when its drawn on paper. Sketching flowcharts will be a fun filled educative method for teaching students about how an algorithm works. Learning by interacting with the system is the best way to learn new topics. This is our motivation behind developing an online flowchart recognition tool which can validate the correctness of the flowchart as well as generate pseudo-code from the connected components in a flowchart.

2 PRIOR WORK

This project deals with hand drawn flowcharts and its conversion to a runnable code to provide interactive feedbacks. There are many related work in this area which uses machine learning to classify the different symbols of a flowchart. [4] uses three layers of identification namely stroke combination, creating candidate symbols using a classifier and then using a graph grammar to identify the logical structure. Another study in the identification of the flowchart is proposed by [2] by using finite state automata. Many related studies have been performed in the area of identification of flowchart components and understanding its grammar. An alternative method is proposed by [10]. They created a system, flowchart designer, which used an iterative method for identifying each component in the flowchart and immediate user feedback is expected if the drawing is wrong. [16] describes two methods to identify offline and online hand drawn flow charts and created a system called FlowChart Editor for online analysis. This system first uses stroke segmentation followed by gesture and connection recognition. [5]

*Both authors contributed equally to this project.

Authors' address: Aswin Jacob Thomas, aswin.jacob.thomas@tamu.edu; Amala Babu, amala1995@tamu.edu, CSCE624 Sketch Recognition, Texas AM University.

have used Conditional random fields and Minimum spanning trees to identify the distance between the strokes and hence to express the underlying contextual information. [2] created a recognition system for online hand

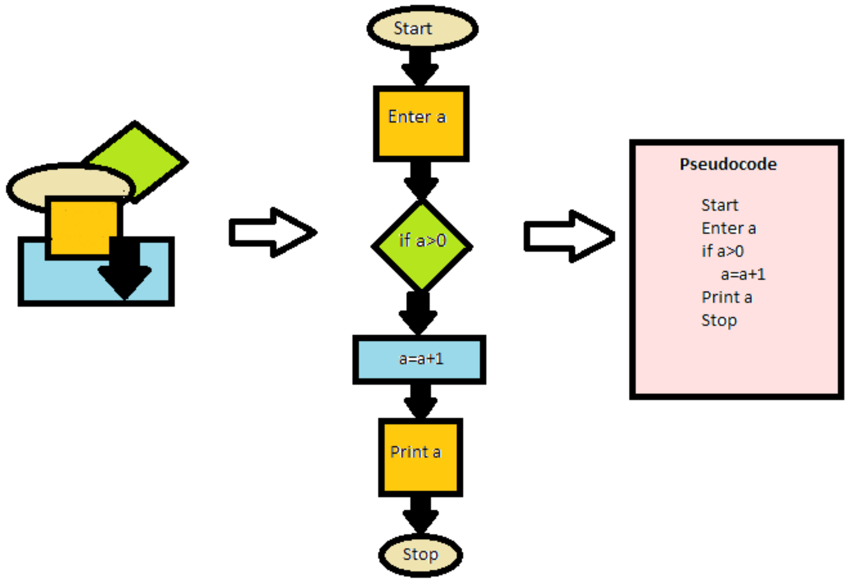


Fig. 1. Methodology

drawn diagrams from domains of arrow connected diagrams, namely flowcharts and finite state automata. They have used the structure of the diagram for recognition as opposed to grammar-based approaches. Miyao [9] segmented the symbols in the flowchart by identifying closed loops real time and then considered the remaining symbols as flow lines. After the identification phase, they also beautified the flowchart and also incorporated handwriting recognition for the text within the components of the flowchart. [7] utilizes a structural method for recognition of symbols in a flowchart. They have improved the existing Description and Modication of the Segmentation method to use a combination of two levels of primitives namely, strokes and line segments. It enables to deal with some difculties previously present because line segments can be used to describe sides of quadrilaterals now. [3] uses both statistical and structural information for flowchart recognition and segmentation tasks. Structural description of symbols is performed using grammar-based methods and statistical description of the same is done by utilizing the geometric properties.

[20] [8] utilized Support Vector Machine and Hidden Markov Model [13] to recognize and convert simple flowcharts into runnable code. We are also trying to perform the same by using feature engineering and graph traversal algorithms. [15] first implemented a system for sketch segmentation followed by sketch recognition and beautification of hand drawn diagrams found in engineering drawings. [6] et. al created a system for recognizing mathematical sketches and free body diagrams called Mathpad using stroke associations and sketch parsing.

3 METHODOLOGY

Our project consists of three main parts for the recognition and interaction with the flowchart. The first stage is to recognize the symbols that make up the flowchart, namely ellipses, rectangles, diamonds, squares and arrows. Ellipses represents the start and end of the control flow in a flowchart. Rectangles are used to represent statements that perform any kind of computation operation. For example, calculating the sum of two numbers will be represented inside a rectangular shape in a flowchart. Diamond symbols are of use in a flowchart to depict conditional statements. Evaluation of a conditional results in one or two branches of control flow that represent different sequences of outcome. Arrows are used to connect the different shapes and maintain the flow of control. For our case, we use lines instead of arrows to represent the control flow. Usually, parallelograms represent input and output operations in a flowchart. Here, for our simplicity, we have used a square symbol in place of parallelogram to represent the same.

After recognizing each symbol in a flowchart, we replace the sketch with a beautified sketch of the recognized symbol. This also serves the purpose of informing the user that the algorithm has identified the sketched symbol. A graph grammar is then constructed in order to store the flow of control of the whole figure. The next step is to generate the pseudo-code of the algorithm that the user represented using a flowchart. Each of the stages is described in detail in each sub-section.

The code is written in JavaScript and HTML, and CSS is used for styling the user interface of the application.

3.1 Symbol Recognition

We have defined five symbols for the flowchart. As mentioned above, they are ellipse, rectangle, diamond, square and straight line. As a first stage of recognition of these symbols, we used $1 template matching algorithm. Templates for each of the symbols were constructed by collecting sample points each symbol from as explained in [18]. The stroke points of a symbol drawn by the user is extracted and compared with the pre-defined templates in the $1 algorithm and the best match is returned. This did not yield satisfying results because the algorithm does not calculate the similarity of the input symbol and the template irrespective of the input symbol's orientation and the order in which the input strokes are drawn. As a result, many of the recognition gave incorrect responses. So, we decided to move forward by using Rubine's [14] features and the features used in Paleosketch [12].

3.1.1 Recognition of Rectangle. A combination of features defined by Rubine [14] and in Paleosketch [12] were used for the recognition of rectangle. Rubine's third feature (f3) - length of diagonal of bounding box, fifth feature (f5) - distance from starting and ending point and eighth feature (f8) - total stroke length were extracted. Another features used for recognition is the length of the major axis. It was calculated by computing the maximum distance between any two points in the stroke. Shortstraw[19] algorithm was used to segment the drawn symbol on the basis of the number of corners detected in it. Evaluation criteria for the symbol to not be a rectangle were:

(1) Ratio of f8 and the perimeter of the bounding box < 1
(2) Number of corners detected < 4
(3) Ratio of f5 and f8 greater than 0.09
(4) Ratio of major axis length and f3 greater than 0.15

If all of the above conditions were false, then the symbol was recognized to be a rectangle.

3.1.2 Recognition of Ellipse. Rubine's third feature (f3) - length of diagonal of bounding box and the length of the major axis of the symbol which calculated by getting the maximum distance between any two points in the stroke was used for the recognition of an ellipse. The first step was to confirm that the input symbol was not a rectangle. Because a rectangle and an ellipse have similar bounding boxes when drawn to the same scale, both ellipse and rectangle share many common features. The algorithm returned true for a symbol whose ratio of the length of the major axis and f3 feature was lesser than 0.95. This is because an ellipse is drawn with its major axis parallel to the X axis in flowchart, which is the same as the longer side of the bounding box. So the length of the major axis will be closer in value to the feature f3. Comparing this ratio will recognize an ellipse in our code, because of the sequence of the checks we have written.

3.1.3 Recognition of Square. The width and height of the bounding box are calculated for the recognition of a square symbol. Also, for a perfectly drawn square, the bounding box will overlap with the sketch. This overlap will be close enough even in the case of an imperfect sketch of a square. In order to utilize this feature, we calculated the average distance from the closest points to each corner of the bounding box. It was empirically found that this value will be lesser than forty for squares. The criteria for a symbol to not be classified as a square are:

(1) The symbol is not a rectangle and value lesser than 1.4 for the ratio of the width and height of the bounding box.
(2) Value greater than 40 for the average distance of closest points to four corners of the bounding box

If all of the above conditions were false, then the symbol was evaluated to be a square.

3.1.4 Recognition of Diamond. Rubine's ninth feature - total curvature of the symbol is calculated for the recognition of a diamond. The algorithm returned false for a symbol whose ratio of f9 and 2π greater than 0.9. As the next step of determining if the symbol was a diamond, we rotated all the points through an angle of $\pi/4$ through the center of the bounding box. The diamond when rotated through $\pi/4$ on the center of its bounding box will resemble a square. So the rotated set of points were passed to the square recognition criteria and if it returned false, the symbol was not recognized as a diamond shaped one, else it is classified as a diamond figure.

3.1.5 Recognition of Line. Rubine's fifth feature (f5) - distance from starting and ending point and eighth feature (f8) - total stroke length were extracted to aid in the recognition of a line. The symbol was recognized as a line if the value of the ratio of f5 and f8 was greater than or equal to 0.95.

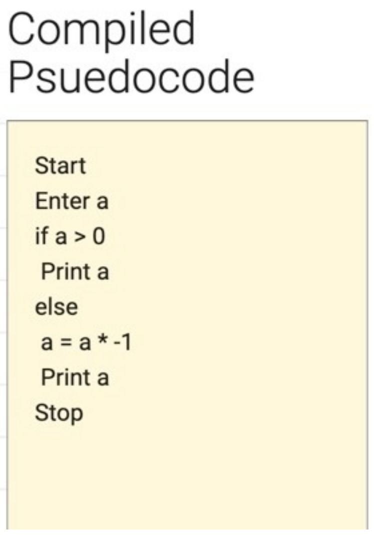
3.2 Redrawing symbols and Construction of Figure

As soon as the user finishes sketching a symbol, it is recognized by the algorithm and the user is asked to enter the text for the symbol. It can be input or output statements for square, computation statements for rectangle and conditional statements for diamond. Start and stop labels for ellipses are handled within the code. The recognized symbols are matched to its counterparts using Paper.js [11] library and redrawn for aesthetic purposes. This also provides the feedback to the user that the figure is recognized by our system. Each figure is redrawn to the scale that the user sketched them and given a color to distinguish their functionality as well as for visual comparison. A recognized line is redrawn as an arrow using the arrow symbol provided by Paper.js library. So after recognition, the flowchart appears with correct symbols to represent flow of control in the diagram.

A graph was constructed to connect and represent the flowchart as a whole. This is necessary in order to perform any operation on the flowchart and convert it into pseudo-code. This process was performed by constructing the flowchart as a connected graph. Each recognized symbol was created as a node in the graph, an object of the class **Figure**. A node has the following attributes which include:

(1) path - Paper.js Path object which represents the corresponding symbol. This path later is used to identify the interaction between the symbols to create the graph.
(2) shape - a string which stores the shape of the symbol. The values taken are line, ellipse, rectangle, diamond and square.
(3) label - a Paper.js PointText object which represents the text written in the symbol. It is placed at the center of each symbol by calculating the center of the bounding box of the symbol.

(4) fromFigure - a Figure object which represents the current node's previous node. This node will have a line drawn from it to the current node.
(5) toFigure - a Figure object which is the next node of the current node. This node will have an arrow drawn from the current node to this node. The last two nodes helps to maintain the structure of the flowchart. These information can be used to create the connections between the nodes.



Compiled Psuedocode

```
Start
Enter a
if a > 0
 Print a
else
 a = a * -1
 Print a
Stop
```

Fig. 2. Code generated from flowchart

3.3 Code Generation

The final stage in the development of the tool is generation of pseudo-code. The graph structure built from the flowchart is utilized for code generation. Pseudo-code begins and ends with “Start" and “Stop" commands. The input text stored in the label attribute of each symbol is extracted and used for code generation. Each figure object is called recursively to extract the input text. Indentation and if/else statements are added in the pseudo-code for conditionals appropriately and the final code is generated.

3.3.1 User Interface Icons. We have added a few buttons in the user interface of the application for the user to easily interact with the developed system and generate the pseudo code. They are:

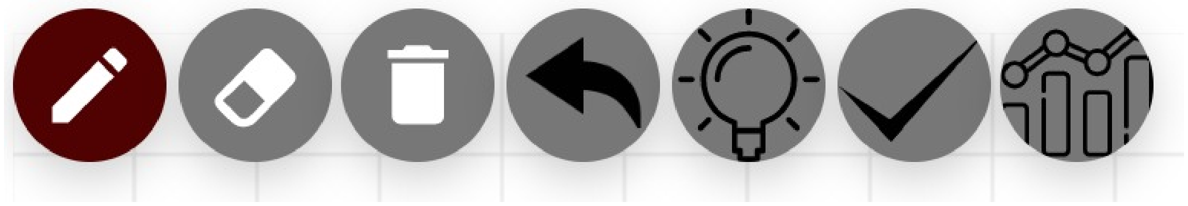
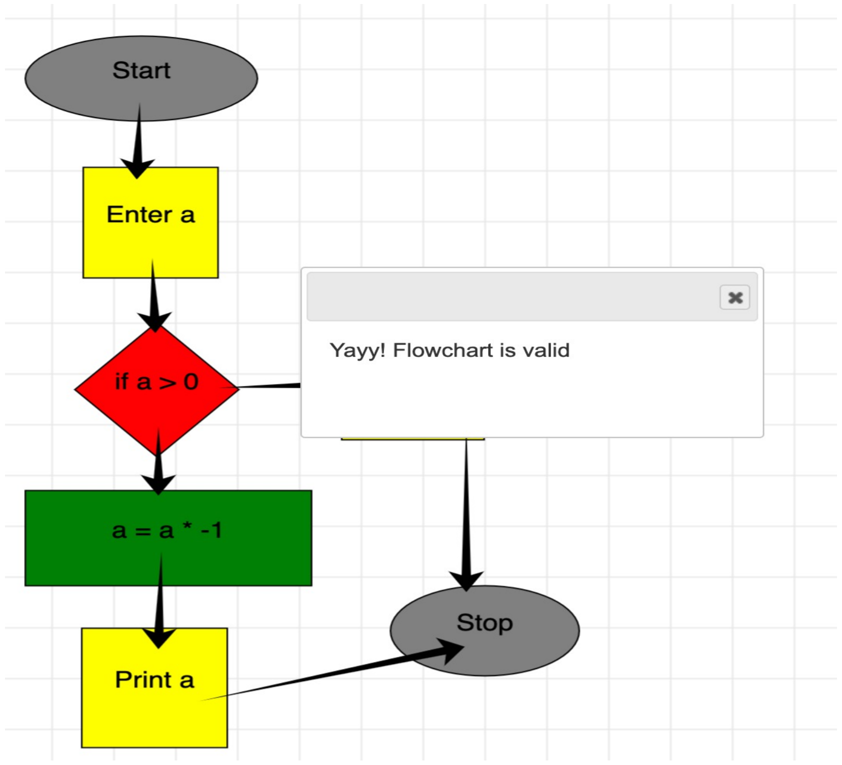


Fig. 3. Buttons for flowchart recognition interface

(1) Undo - Undo button is added so that the user is given the chance to edit the symbol if he made a mistake or if the algorithm didn't recognize the symbol as he intended to. Undo button is used for penalizing the intended figure as it did not get detected in the first trial, hence a false negative input. This is also treated as a false positive input for the incorrectly recognized symbol. Pressing this button removes the node from the window.figures object which stores all the drawn figures so that its removal is taken into effect from the whole figure in real time.
(2) Generate Pseudo-code - pressing this button generates the pseudo-code on the right side of the screen
(3) isValid - the validity of the user sketched flowchart is checked using the isValid button. Depth first search is performed on the graph structure for the implementation of the validity check of the flowchart. The conditions checked are:
  (a) return false if the user hasn't sketched any figure
  (b) return false if the flowchart doesn't start with the start symbol
  (c) return false if the start symbol has more than one arrow extending from it
  (d) return false if any symbol other than ellipse and diamond has more than two arrows extending from it.
  (e) return false if diamond has more than two arrows extending from it
  (f) return false if diamond has more than one arrow extending to it
  (g) return false if all paths in the flowchart eventually does not end in stop symbol

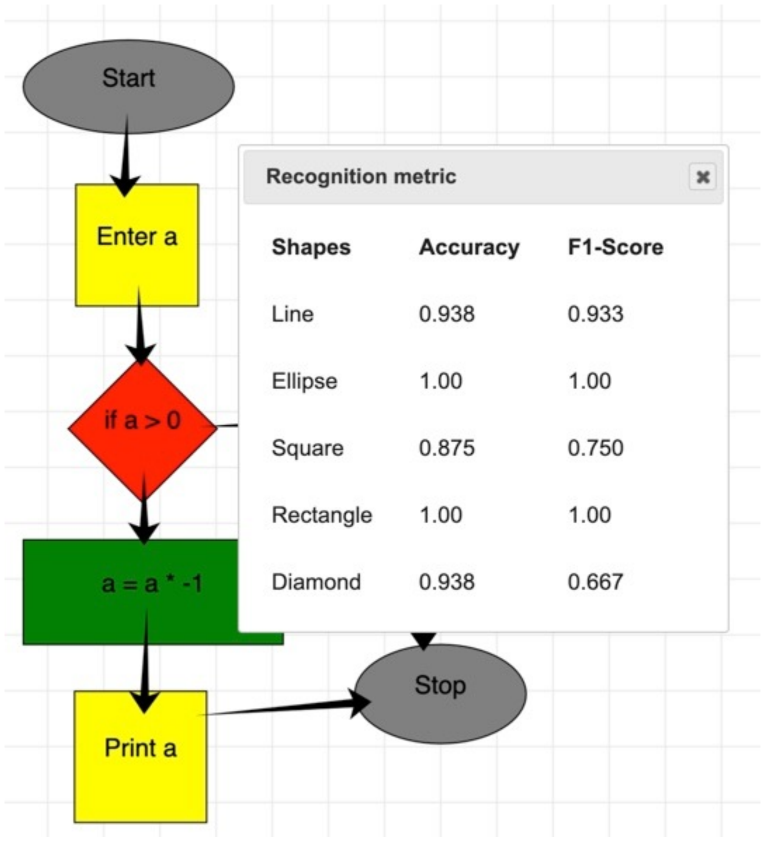


Yayy! Flowchart is valid

Fig. 4. A valid flowchart recognition

(4) Show Metrics - pressing this button displays the evaluation metrics of the symbol recognition until now. Accuracy and F-measure values are displayed for each of the symbol. A correct recognition of a symbol is

counted as true positive. An incorrect recognition is considered as false negative and false positive for the symbol which was incorrectly displayed and the count of true negative is incremented by one for all other symbols. On the click of undo button, the system penalizes both the intended symbol and the obtained symbol even if the user changed his mind while drawing, hence we use an aggressive penalizing technique.



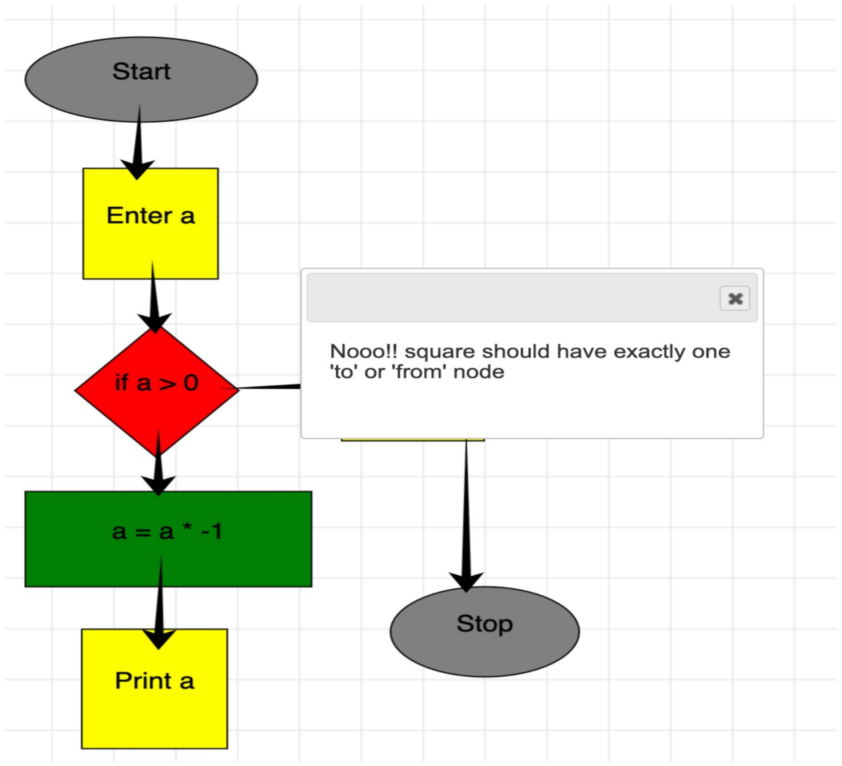
Shapes	Accuracy	F1-Score
Line	0.938	0.933
Ellipse	1.00	1.00
Square	0.875	0.750
Rectangle	1.00	1.00
Diamond	0.938	0.667

Fig. 5. Evaluation Metrics generated from flowchart

4 EVALUATION

We provide the feature to see the evaluation metric for recognizing each symbol for the current session of the drawing. After each session, if the browser is refreshed or closed, the metric will be reset. This will make sure that the metric is user independent and purely depends on the way the user draws the symbols. We created a new class called Metric.js for evaluating the flowchart recognition tool. An object of the Metric class has the following properties:

(1) shape - defines the shape of the symbol. It takes one of the following values - ellipse, line, square, rectangle or diamond
(2) TP - stores the number of times the symbol was recognized correctly, as the intended symbol
(3) TN - stores the number of times the symbol was not recognized incorrectly



Nooo!! square should have exactly one 'to' or 'from' node

Fig. 6. An invalid flowchart recognition

(4) FP - stores the number of times the symbol was recognized in place of the correct intended symbol
(5) FN - store the number of times the intended symbol was not recognized and an incorrect symbol was outputed.

The evaluation was performed for the following cases:

(a) When Undo button is pressed - this happens either when the user has made a mistake in the diagram or if the algorithm has incorrectly recognized the symbol. Both these cases are penalized and the false positive count of the incorrectly recognized symbol and the false negative count of the missed symbol is incremented
(b) When the system throws the Cannot identify the symbol pop-up - this case is penalized because this is a shortcoming of the algorithm to recognize what the user has sketched. The false negative count of the intended symbol is incremented when the user draws it next.
(c) When the cancel button is pressed when the user is asked to input the text to be associated with the symbol - This case also happens when the user has made a mistake in the diagram or if the algorithm has incorrectly recognized the symbol. Both these cases are penalized and the false positive count of the incorrectly recognized symbol and the false negative count of the missed symbol is incremented

The results obtained by our application are shown in the table.1 Accuracy and F-measure are used as the evaluation metrics. Accuracy is calculated as

$$\text{Accuracy - } \frac{TP + TN}{TP + TN + FP + FN}$$

Table 1. Evaluation Metrics

Symbol	Accuracy	F-measure
Ellipse	0.908	0.755
Square	0.951	0.873
Rectangle	1	1
Diamond	0.908	0.581
Line	0.972	0.968

and F-measure is calculated as

$$\text{F-measure - } \frac{2TP}{FP + 2TP + FN}$$

As we can see, from the table, diamond symbol obtained the least F-measure score because the recognition algorithm used for diamond is not strong enough to distinguish between diamond and ellipse. Hence, this phenomenon also increased the false positive count of ellipse which in turned decreased its F-measure. The accuracy for all symbols is above 0.9 while F-measure varies between 0.5 and 1. This is because, every time a symbol is correctly recognized, the true negative count for all other symbols is incremented by one. Hence, this brings out the inherent flow in accuracy in comparison with F-measure.

5 FUTURE WORK

Flowcharts are an effective tool to teach the basics of writing algorithms and understanding the flow of logic in them. They are a fun way to learn logic and solve problems which will not tire the students. They help in displaying and storing information intuitively which can be easily understood by even a novice.

As a future work, we plan to convert the pseudo-code into runnable code which can take handwritten inputs, recognize those inputs, perform computation and show the result to the user. We can also utilize $N [1] and $P [17] algorithms for recognition instead of feature engineering to improve the recognition rates. $P uses point cloud method while $N generalizes multistrokes to find the perfect match. We also plan to provide interactive feedback to the user at each stage of the flowchart so that the logic behind the drawing can easily be understood. Currently we support only computation, conditional and input/output statements in our flowchart and we would like to extend it to include looping statements as well.

6 ACKNOWLEDGMENTS

We would like to thank Professor Tracy Anne Hammond for her motivation and valuable feedback throughout the duration of the course. This greatly inspired us to work on this project and take it to a successful completion. Her enthusiasm for the subject has motivated us to push our boundaries and learn more.

REFERENCES

[1] Lisa Anthony and Jacob O Wobbrock. 2012. $N-protractor: a fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*. Canadian Information Processing Society, 117–120.

[2] Martin Bresler. 2016. Understanding Formal Arrow-Connected Diagrams and Free-form Sketches. (2016).

[3] Cérès Carton, Aurélie Lemaitre, and Bertrand Coüasnon. 2013. Fusion of statistical and structural information for flowchart recognition. In *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 1210–1214.

[4] Quan Chen, Dapeng Shi, Guihuan Feng, Xiaoyan Zhao, and Bin Luo. 2015. On-line handwritten flowchart recognition based on logical structure and graph grammar. In *2015 5th International Conference on Information Science and Technology (ICIST)*. IEEE, 424–429.

[5] Adrien Delaye. 2014. Structured prediction models for online sketch recognition. *interpretation* 1, 3 (2014), 4–16.

[6] Joseph J LaViola Jr and Robert C Zeleznik. 2004. MathPad 2: a system for the creation and exploration of mathematical sketches. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 432–440.

[7] Aurélie Lemaitre, Harold Mouchère, Jean Camillerapp, and Bertrand Coüasnon. 2011. Interest of syntactic knowledge for on-line flowchart recognition. In *International Workshop on Graphics Recognition*. Springer, 89–98.

[8] Elvis Wai Chung Leung, Fu Lee Wang, Lanfang Miao, Jianmin Zhao, and Jifeng He. 2008. *Advances in Blended Learning: Second Workshop on Blended Learning, WBL 2008, Jinhua, China, August 20-22, 2008, Revised Selected Papers*. Vol. 5328. Springer.

[9] Hidetoshi Miyao and Rei Maruyama. 2012. On-line handwritten flowchart recognition, beautification and editing system. In *2012 International Conference on Frontiers in Handwriting Recognition*. IEEE, 83–88.

[10] Sachi Mizobuchi and Michiaki Yasumura. 2004. Tapping vs. circling selections on pen-based devices: evidence for different performance-shaping factors. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 607–614.

[11] paperjs [n.d.]. Paper.js. http://paperjs.org/reference/global/.

[12] Brandon Paulson and Tracy Hammond. 2008. Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces*. ACM, 1–10.

[13] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

[14] Dean Rubine. 1992. Combining gestures and direct manipulation. In *Chi*, Vol. 92. 659–660.

[15] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. 2001. Sketch based interfaces: early processing for sketch understanding. In *Proceedings of the 2001 workshop on Perceptive user interfaces*. ACM, 1–8.

[16] Wioleta Szwoch and Michał Mucha. 2013. Recognition of hand drawn flowcharts. In *Image Processing and Communications Challenges 4*. Springer, 65–72.

[17] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O Wobbrock. 2012. Gestures as point clouds: a $P recognizer for user interface prototypes. In *Proceedings of the 14th ACM international conference on Multimodal interaction*. ACM, 273–280.

[18] Jacob O Wobbrock, Andrew D Wilson, and Yang Li. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 159–168.

[19] Aaron Wolin, Brian Eoff, and Tracy Hammond. 2008. ShortStraw: A Simple and Effective Corner Finder for Polylines.. In *SBM*. 33–40.

[20] Zhenming Yuan, Hong Pan, and Liang Zhang. 2008. A novel pen-based flowchart recognition system for programming teaching. In *Workshop on Blended Learning*. Springer, 55–64.