On-line handwritten flowchart recognition, beautification, and editing system

Hidetoshi Miyao and Rei Maruyama
Faculty of Engineering, Shinshu University, Japan
miyao@cs.shinshu-u.ac.jp

Abstract

In order to segment and recognize on-line handwritten flowchart symbols precisely, we propose a method that segments the graphic symbols based on the loop structure and recognize the segmented symbols by using SVMs. In our experiments, low error rate of 3.37% for symbol segmentation and high recognition rate of 97.6% were obtained. We also propose a beautification and editing method for recognized symbols, and implement them to construct a prototype system. We compare an input time for drawing flowcharts between our system and a traditional application using icon-based interface. As a result, the input time on our system was faster than that on traditional one for flowcharts without texts.

Keywords: flowchart recognition, symbol beautification, on-line handwritten symbol, segmentation, SVM.

1. Introduction

A flowchart is a diagram that can represent algorithms or process flow by using symbols of boxes and flow lines. The shape of the box is determined according to the type of process and a text in the box represents the concrete content of the process. The flow lines connect with the boxes and can represent the order of processes.

Users often use commercial applications such as Microsoft Visio [1] when they draw flowcharts. In such systems, symbol icons are located beside a drawing area, a user selects a desired icon from them, and drags it to the drawing area with a computer mouse. A flowchart can be completed by these operations repeatedly. However, these systems have the following disadvantages:

- It should be a troublesome task to draw a symbol because a user has to move a pointing device frequently over a long distance between the symbol icon area and the drawing one.
- The symbol icon area narrows the region of drawing area.

Therefore, several on-line handwritten flowchart recognition systems with a pen computer have been developed [2,3,4]. In the systems, the main problem is that it could not segment flowchart symbols precisely. Yuan et al. [2] have realized to segment between graphic symbols (i.e. boxes and flow lines) and texts by forcing a user to select an input mode explicitly. Moreover, the graphic symbols are segmented based on the operation rule that one graphic symbol must be drawn with one stroke. However, it would be inconvenience for some graphic symbols since it is unnatural to draw it with one stroke. Lemaitre et al. [4] have proposed the method that can segment the symbols by defining syntactic rules for flowchart even if the graphic symbols and texts are written all together and each graphic symbol is drawn with one or more strokes. However, the result of text segmentation and recognition rate is 71.7% and the rate for graphic symbols is 72.8%, it would not be enough for practical use.

In order to obtain high performance of segmentation and recognition of flowchart symbols, we propose a method with the following features:

- For segmentation between graphic symbols and texts, we have considered that the previously proposed methods could not realize to obtain enough segmentation performance. Thus we adopt the method that a user can add a text to a recognized graphic symbol by using another input window. The text can be handwritten and then recognized. In this way, it would be easy to write texts and they can be segmented precisely.
- We have noticed that graphic symbols except flow lines include a closed loop. Thus, we

978-0-7695-4774-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICFHR.2012.250

propose a method that a closed loop is checked every time a stroke is drawn, if the closed loop is extracted, then it is segmented as a graphic symbol. In this way, they can be segmented without complex structural syntactic rules. Moreover, a user can draw a graphic symbol with one or more strokes which can be written in any order.

In the related works [2,3,4], methods for beautification of symbols and editing them are not mentioned. Therefore, we also propose how to fix up recognized symbols serially and edit them. In our system, the edit function includes movement, resizing, deletion, and adding a text for graphic symbols.

2. Target symbols and system constraints

We assume that a flowchart diagram is drawn with a pen device on a tablet PC or a tablet device connected to a PC. The target 12 symbols are shown in Fig.1. We refer to the symbols from #1 to #11 as *graphic symbols*. In particular, the symbols from #1 to #10 are referred to as *loop symbols*. The symbols #1, #2, #3, #4, #5, #11, and #12 are treated in the related researches [2,3,4], i.e. the symbols from #6 to #10 are not treated.

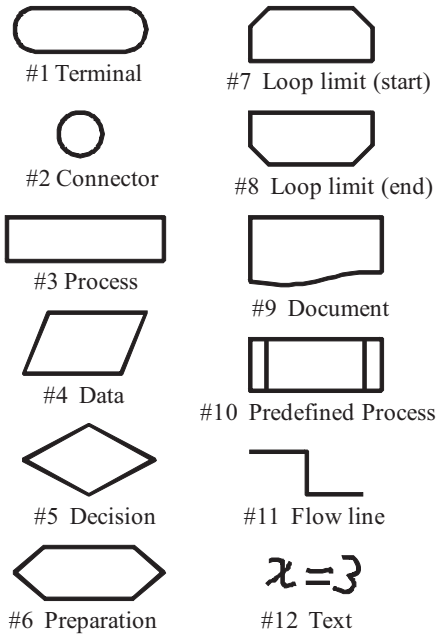

Figure 1. Target symbols

The system constraints are as follows:

- Strokes which belong to a loop symbol, must be drawn in succession. But they can be drawn in any order.
- A stroke which belongs to two or more graphic symbols is not permitted. For example, flow line and decision symbol cannot be written with one stroke.
- Symbol #10 has to be written with one stroke because only this symbol includes a multi-loop structure.
- Interflow of flow lines is not permitted. In other words, terminal points of flow lines must connect with either a graphic symbol or nothing.

3. Symbol segmentation and recognition

We assume that each stroke is a connected component from pen-down to pen-up. We also assume that each stroke is represented as a sequence of 2D coordinates of pen positions. In order to divide handwritten strokes into strokes which belong to loop symbols (from #1 to #10 in Fig.1) and others which belong to flow lines, we use the following procedures:

- Terminal points of unfixed strokes are examined every time a stroke is drawn.
- If the positions of any two terminal points are close enough, they are connected.
- It is examined whether there is a loop structure. If the loop structure is extracted, a set of unfixed strokes (or one unfixed stroke) which belong to the loop, is segmented as a loop symbol and other strokes are extracted as flow lines (as shown in Fig.2).

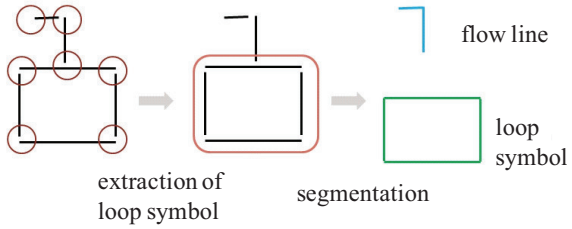

Figure 2. Loop symbol segmentation

Next step is to recognize the segmented loop symbol. The bounding box that surrounds a sequence of sampling points of the loop symbol is normalized to 64 x 64 pixels. An image of the loop symbol is generated by connecting the sequence of points for each stroke. The loop symbol image is then partitioned into 8 x 8 blocks. Four patterns emphasizing four directions (vertical, horizontal, left slant, and right slant) at every block are detected. As a result, 256 features (8 x 8 x 4) are extracted. This feature does not depend on the size of handwritten symbol and the stroke order. To classify the loop symbols, we use SVM (a two-class classifier) [5]. For an individual

class, an SVM is trained using the features extracted from all the test examples and the corresponding target values (1 for positive examples and -1 for negative). However, after the size normalization process, it is difficult to distinguish between symbol #1 (Terminal) and #2 (Connector) because they are almost the same image. Therefore, we treat them as the symbols that belong to a same class in this step. As a result, 9 SVMs (i.e. SVMs for classifying each loop symbol #1+ #2, #3, #4, ..., #10) are constructed. In practice, we adopted the Gaussian kernel as a kernel function and used SVM^{light} [6] to train the SVMs. To apply SVM to a multiclass symbol recognition problem, we used the one-versus-the-rest (1vr) approach [5]. Finally, if the recognized class is symbol #1+#2, we can classify them based on the aspect ratio of the bounding box of the symbol. In this way, the segmented loop symbol is recognized.

The recognized loop symbol is reshaped based on the position and size of original handwritten symbol and the type of the recognized class. Then it is displayed on the drawing area. At the same time, the candidates of loop symbols are displayed aside of the reshaped symbol as shown in Fig.3. They are ordered according to the output value of SVMs. A user can select a symbol from the candidates and modify the recognition result if necessary.

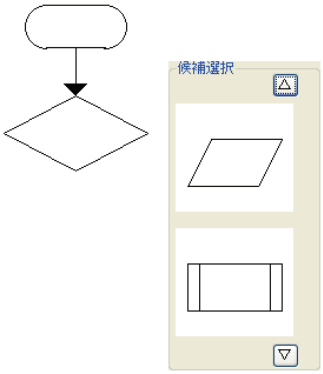

Figure 3. Candidate selection of loop symbols

4. Beautification of recognized symbols

Next process is to make a connection between associated symbols, fix up them and output them to the drawing area. This process is done every time after a loop symbol is recognized. As a result, all the symbols come to be fixed. The process is as follows:

(1) If a terminal point of a flow line is close enough to a loop symbol (i.e. the terminal point is located in the dotted box surrounded the loop symbol shown in Fig.4), the terminal point is connected to the nearest point among 4 black dots on the edge of loop symbol as shown in Fig.4. In this connection, if the line flows in the loop symbol, an arrow symbol is attached to the terminal automatically. The flow direction is determined based on the writing direction of the flow line.

(2) If a terminal point of a flow line is close enough to a terminal point of another flow line, they are connected with each other.

(3) Flow lines are reshaped with horizontal line and vertical one. If a flow line intersects with a loop symbol, the line is also reshaped to avoid the intersection.

(4) If two or more loop symbols are placed almost vertically, all the symbols are aligned with the horizontal center line of the top loop symbol.

Figure 4. Connection between flow line and loop symbol

Flow lines written after drawing a loop symbol still remain in unfixed situation in the drawing area. To fix them explicitly by a user, when the side button of a pen device is pressed, the above mentioned process is also invoked.

5. Editing functions

A user can apply an edit function to fixed symbols. This function can be called by pressing an objective symbol or the other drawing area for 1 second with the pen device and then a menu is popped up around the cursor as shown in Fig.5. The user can select the intended function from the menu and apply it to the objective symbol. The contents of the menu depend on the type of selected object as denoted in Table 1.

pressing the symbol for 1 second

menu is popped up

Figure 5. Calling pop-up menu for editing

Table 1. Contents of pop-up menu for each object

Type of object	Content of operations
Loop symbols	Deletion, Adding a text, Modification of symbol, Display of property
Flow lines	Deletion, Adding a text, Reverse of arrow direction, Display of property
Other regions	Deletion of all symbols, Display of property

If the user selects the function of adding a text for loop symbols, another window is popped up and the user can handwrite texts there as shown in Fig.6. They are recognized by the handwriting recognizer developed by Microsoft [7] and it is pasted into the objective symbol. On the other hand, if the user selects the function of adding a text for flow lines, the user can only select a text of either "Yes" or "No" in the current system. The text is pasted aside of the objective line.

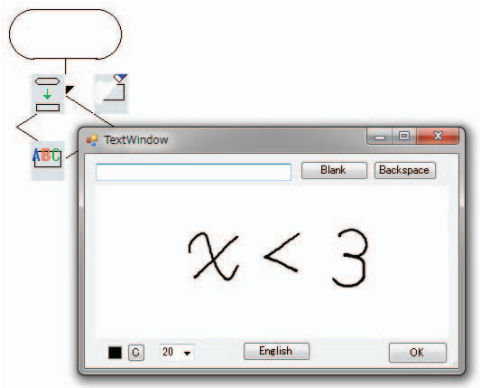

Figure 6. Handwriting recognizer for text inputs

In order to move or resize a fixed loop symbol, the user has to point the symbol and press the side button of the pen device, and then the 4 vertices of bounding box for the loop symbol are indicated by red spheres as shown in Fig.7. In this situation, if the pen is moved with pressing the button (i.e. it means a drag operation), the symbol can be moved. On the other hand, the drag operation for one of the red spheres is to resize the symbol. In these operations, the flow lines connected with the edited symbol are reshaped as shown in the right of Fig.7. Furthermore, if the flow lines intersect with a loop symbol, the lines are also reshaped without the intersection.

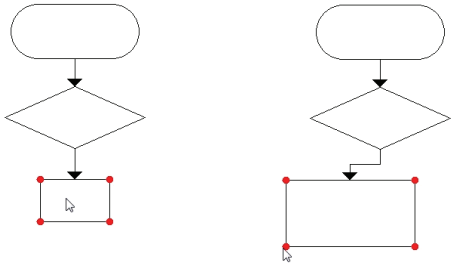

Figure 7. Resizing loop symbol

6. Experimental results

In the experiment, we used a tablet device (WACOM Intuos3 PTZ-630) connected to a PC.

First, we explain an experiment for symbol segmentation. We used 5 flowcharts shown in Fig.8 for the evaluation. Two writers wrote them (except for texts) 10 times for each sample. As a result, the error rate of symbol segmentation was 3.37% (=35/1040). The reason of the error was that some loop symbols could not be structured when two terminal positions of strokes were away from each other and they were not connected in despite of they should be connected ideally.

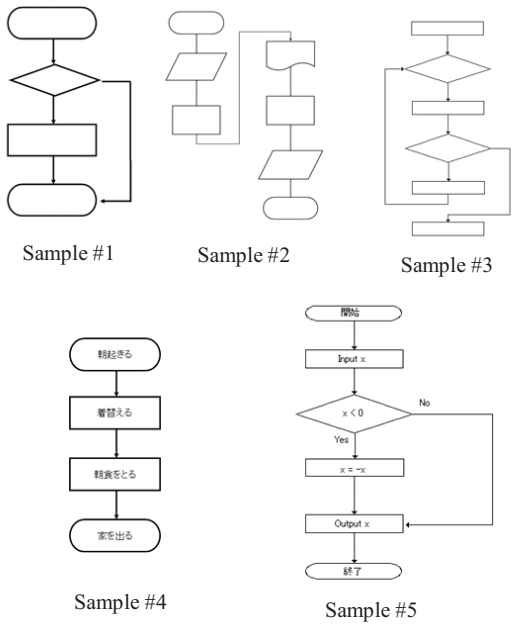

Figure 8. Samples used for experiments

Table 2. Results of symbol recognition

Symbol type	Recognition rate [%]
#1 Terminator	92.0
#2 Connector	96.5
#3 Process	99.0
#4 Data	98.5
#5 Decision	100.0
#6 Preparation	98.0
#7 Loop limit (start)	97.5
#8 Loop limit (end)	95.0
#9 Document	99.5
#10 Predefined process	100.0
Total	97.6

Next, we examined the performance of symbol recognition assuming that the symbols are correctly segmented. A writer wrote 120 symbols for each loop symbol (from #3 to #10) and wrote 60 symbols for each of Terminal (#1) and Connector (#2) symbols. In total, 1,080 samples were acquired and they were used to train 9SVMs. For other two writers, each writer wrote 100 symbols for each symbol (from #1 to #10). In total, 2,000 samples were acquired and they were used as test samples. Table 2 shows the results of symbol recognition rates for the test samples. The high average recognition rate of 97.6% was obtained. From these results, the error rate of symbol segmentation is low (3.37%) and the symbol recognition rate is high (97.6%), therefore the total accuracy should be more than 90%. In the related work [4], the result of graphic symbol segmentation and recognition rate is 72.8%. We could not simply compare this result with ours because the previous work deal with symbols including texts and the target symbols are different. However, it is considered that the symbol segmentation and recognition rate of our system is fairly high.

Finally, in order to evaluate the usability of our system, we compared our system with the commercial application Microsoft Visio 2010 [1].

Regarding the space of drawing area, the area of our system is about 10% larger than that of Visio since our system does not have to locate a various kinds of icons. Therefore, it could be more suitable for use on a portable terminal with restricted drawing area.

Next, 12 writers wrote 5 flowcharts shown in Fig.8 using both systems and the input time was measured. Sample #4 and #5 include English and Japanese texts. Table 3 shows the average input time for each sample. From these results, the input time of our system is slightly faster than that of Visio only if graphic symbols are drawn. The reason of obtaining only little difference is that almost writers are unfamiliar with a tablet device. Thus, the same 5 flowcharts without texts were drawn by another writer who was familiar with a tablet. As a result, the total input time was about 122[s]. It is much faster than the total average time (158.47[s]) shown in Table 3. Therefore it is considered that the input time can be greatly improved by being familiar with a tablet device. Moreover, in Visio, the movement distance of the pointer tends to be long since a user has to move the pointer frequently between the icon area and the drawing area. On the other hand, in our system, most of the operations can be done without large movement of the pointer. This is the reason why the input time of graphic symbols with our system is faster than that with commercial applications. However, regarding the input time of symbols including texts (samples #4 and #5 in Fig.8), the input time of our system is fairly longer than that of Visio since an input time for handwriting texts is generally longer than that for writing texts with keyboards and it also takes a little time to call the edit menu in our system.

Table 3. Average input time for each sample [s]

System	#1	#2	#3	#4	#5
Ours	28.33	41.51	42.30	78.24 (17.10)	134.33 (29.23)
Visio	31.27	43.18	43.48	40.14 (14.95)	82.71 (31.95)

The values in parentheses denote the time not included time of text input

7. Conclusion

In order to segment and recognize on-line handwritten flowchart symbols precisely, we proposed the method that segments the graphic symbols based on the loop structure and recognize the symbols by feeding the directional features of symbol image into SVMs. In our experiments, the error rate of segmentation was 3.37% and recognition rate was 97.6% for the 10 target symbols. The results show that our system is sufficiently practical. However, there are the following problems in the current system:

- The user always requires attention to making a loop structure when he/she draws a graphic symbol.
- Loop symbols including a multi-loop structure have to be written with one stroke. Thus, it is unnatural way to draw.

We also proposed the beautification and editing methods for recognized symbols, and implemented them to construct a system. In order to examine the

usability of our system, we compared it with a traditional application using icon-based operations. As a result of the experiment, the input time on our system was faster than that on the traditional one for only graphic symbols. However, it took a longer time to input texts in our system.

In future work, we plan to improve the system by solving the above mentioned problems

References

[1] Microsoft Visio 2010, http://office.microsoft.com/en-us/visio/

[2] Z. Yuan, H. Pan, and L. Zhang, "A Novel Pen-Based Flowchart Recognition System for Programming Teaching," in WBL, ser. Lecture Notes in Computer Science, E. W. C. Leung, F. L. Wang, L. Miao, J. Zhao, and J. He, Eds., vol. 5328. pp.55-64, Springer, 2008.

[3] A. M. Awal, G. Feng, H. Mouchere, and C. V. Gaudin, "First Experiments on a new Online Handwritten Flowchart Database," in Document Recognition and Retrieval XVIII, 2011.

[4] A. Lemaitre, H. Mouchere, J. Camillerapp, and B. Couasnon, "Interest of syntactic knowledge for on-line flowchart recognition," in Proc. of ninth IAPR International Workshop on Graphics Recognition (GREG2011), pp.85-88, 2011.

[5] C. M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006.

[6] T. Joachims, "Making large-scale SVM learning practical," In Advances in Kernel Methods, Chapter11, MIT Press, 1999.

[7] Microsoft MSDN, http://msdn.microsoft.com/