A Recognizer for Free-Hand Graph Drawings[1]

Hamdi Dibeklioglu
Bogazici University
Dept. of Computer Eng.
Istanbul, Turkey
Hamdi.Dibeklioglu@boun.edu.tr

Tevfik Metin Sezgin
University of Cambridge
Computer Laboratory
Cambridge, UK
Metin.Sezgin@cl.cam.ac.uk

Ender Ozcan
Yeditepe University
Dept. of Computer Eng.
Istanbul, Turkey
eozcan@cse.yeditepe.edu.tr

Abstract

Interactive multimedia such as computer simulations and animations received increased attention over the years as supplementary teaching tools and have now become integral components of most engineering and science curriculums. We believe one way to boost the utility of such simulations and animations is to make them easier to use. In this paper, we describe a pen-based interface for constructing weighted and unweighted graphs which functions as a front-end to a shortest path and minimum spanning tree (MST) algorithm simulators that we have developed. We compare the usability of this pen-based interface to that of a WIMP-based interface and a hybrid interface that uses a mixture of pen-based and WIMP-based input methods. We report evaluation results on the usability of the three types of interfaces that we have developed. Our results show that a purely pen-based approach to graph creation may actually suffer from the relatively higher misrecognition rates of the harder-to-recognize symbols and inhibit usability. We conclude that a hybrid approach which combines a soft-keyboard with graph recognition outperforms interfaces that are purely pen-based or WIMP-based. [1]

1. Introduction

Interactive multimedia such as computer simulations and animations have moved beyond being supplementary teaching tools in education and now become integral components of most engineering and science curriculums.

Although there is a wealth of resources for interactive animations, these applications usually employ traditional WIMP-based interfaces regardless of the appropriateness of such interfaces for the task at hand. For example, a quick web search for shortest path graph algorithms returns over a hundred applets/applications with WIMP-based interfaces despite the fact that drawing a graph – the first step of interaction for all these applications – could naturally be done using a pen-based interface due to its diagrammatic nature.

Pen-based interfaces are not only better suited for this task, but also make it possible to interact with the application in question in a relatively standard diagrammatic language unlike their WIMP counterparts. For example, quick examination of the WIMP-based shortest-path animation applications reveals that they show considerable variation on how the user is expected to construct the graphs. For adding nodes to the graph some applications use buttons with images, text, or both. Some use pull-down menus, some use a drag-and-drop interface while others use a combination of choice menus and buttons. Textual labels associated with the GUI elements also show variation. For example, the words "node/vertex" and "edge/link/arc/line/connector" are used interchangeably although they conceptually refer to the same thing and have standard graphical/visual representations.

The above observations give us reasonable support to believe that an intelligent pen-based interface which interprets free-hand pen input would be preferred over traditional WIMP-based interfaces for constructing weighted/unweighted and directed/undirected graphs. The usefulness of such a system will depend on many factors including the degree to which we can correctly interpret free-hand ink, the perceived ease-of-use for the users, the efficiency of the input method and the overall satisfaction of the users. To answer these questions, we have built three interfaces for constructing graphs:

1. *a pen-based interface* where the graph structure and weights are entered using free-hand ink
2. *a WIMP-based PowerPoint-like interface* to construct the graph and a soft-keyboard to enter edge weights
3. *a hybrid interface* using pen input for graph construction and a soft-keyboard for inputting edge weights

All of these three interfaces function as front-ends to a shortest path algorithm simulator that we have developed.

[1] More details on this work is available at:
http://people.csail.mit.edu/mtsezgin/publications.htm

In the rest of this paper, we describe the architecture of our application including a summary of each front-end interface in detail. We also report evaluation results on the usability of each interface. Our results show that the benefits of a purely pen-based approach to graph creation may actually be undermined by the relatively poor recognition rates of the harder-to-recognize symbols and inhibit usability. We conclude with a discussion of the related and future work.

2 Application Architecture

We have designed our application to have a clear separation between the graph construction module and the algorithm animation module. We use a graph adjacency matrix representation to interface the front-end and the animation modules. This makes it easy to extend the system by adding a new input method or a new animation module.

2.1. Graph Construction Front-Ends

2.1.1 Ink recognition method

The ink-based graph construction method has three components that respectively support graph structure recognition, digit recognition, and editing.

Recognizing the graph structure
The graph structure is constructed by recognizing circles and lines (for nodes and edges) and points (for entering the edge-weight editing mode). We only support single stroke objects. So we assume that the users draw nodes and links using single strokes, and also write digits in single strokes.

Node and edge recognition
In our pen-based interface, nodes and edges are recognized by fitting circles and lines to the input stroke, and choosing the interpretation that has a lower least squares error.

To find a line fit, we use the first and last points in the stroke, (x_1, y_1) and (x_n, y_n), to derive the line equation $ax + by + c = 0$ using the following equation of a line with two known points: $(y - y_1)(x_n - x_1) = (y_n - y_1)(x - x_1)$.

To fit a circle, we find the rectangular bounding box with corner positions (x_{min}, y_{min}) and (x_{max}, y_{max}) for the stroke at hand where $x_{min}, y_{min}, x_{max}$ and y_{max} are the minimum and maximum values for the x and y positions of the points in the stroke. Then the equation $(x - h)^2 + (y - k)^2 = r^2$ gives us the circular fit to the stroke where (h, k) is the center of the circle and r is the radius and $h = \frac{x_{min}+x_{max}}{2}, k = \frac{y_{min}+y_{max}}{2}$.

After the line and circle fits are computed, we compute the distance of each point (p_x, p_y) in the stroke to the line and circle fits using the following error functions:

$$Error_{line}(p_x, p_y) = \left| \frac{ap_x + bp_y + c}{\sqrt{a^2 + b^2}} \right|,$$

$$Error_{circle}(p_x, p_y) = \sqrt{(h - p_x)^2 + (k - p_y)^2} - r.$$

We compute the average fitting error for circle and line fits using all points in the stroke and classify it to have the label of the fit with the lowest average error. Circles indicate that the user has drawn a node, and lines indicate an edge. If the end points of the stroke lie on previously drawn nodes, then those nodes become the source and destination nodes. If the user is drawing a directed graph, the direction of the stroke also determines the direction of the edge. If either of the stroke's end-points don't lie inside a node, then the stroke is ignored.

If the recognized shape is a circle that surrounds a set of nodes, then it is interpreted as a selection operation and passed onto the graph editing module.

Handwritten digit recognition
Digit recognition allows the edge weights to be specified by pen input. We have implemented two digit recognition methods. The first one uses Kohonen networks [5]. The other recognition method is based on the iterative closest point [3] and parallel sampling algorithms, and we describe it in detail in [1]. The method based on Kohonen networks has outperformed the latter algorithm and it was our algorithm of choice for the version of the system used in the evaluation.

In addition to digits, we have trained the system to recognize two simple gestures: a leftward dash deletes the last digit that was recognized, and a checkmark signals that the user is done entering the current weight.

The users can edit their graphs using the pen interface. Nodes are selected by drawing a closed loop around them. Once the nodes are selected, the selection can be dragged around to move the nodes. Drawing a line across the selection deletes the selected nodes and any edges connected to them. Tapping the weight of an edge with the stylus puts the system into edge-weight editing mode where the user can update the weight value using the pen interface.

2.1.2 Hybrid Method

The hybrid method uses the graph recognition method described above for specifying the graph structure. We use a soft-keyboard for entering the edge weights, which works much like the soft-keyboard included with the Windows XP Tablet PC version. The user taps in the keys with a stylus or another pointing device to enter the edge weights. Editing in the hybrid method works the same way as in the ink recognition method, except the edge weights are entered using the soft keyboard.

2.1.3 WIMP-based method

Fig. 1 shows our WIMP based interface. The WIMP based method uses image-buttons with textual labels for constructing the graph structure. The image-buttons form a toolbar, and the user selects the operation that needs to be performed by clicking on the appropriate button in the toolbar. This interface is representative of most WIMP-based

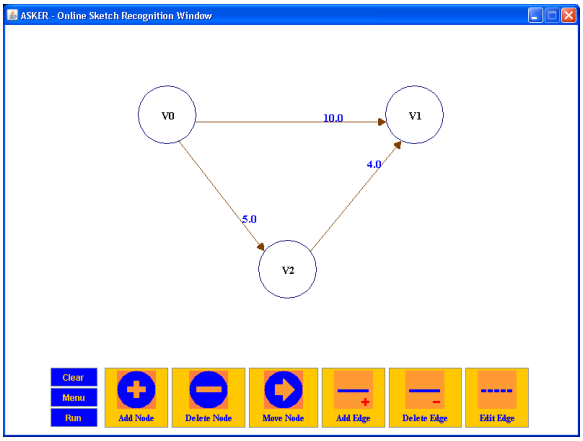

Figure 1. The WIMP-based interface.

tools such as MS PowerPoint. As it was the case for the hybrid method, the edge weights are entered using the soft-keyboard interface. The graph editing operations are selected using the toolbar.

2.2 Graph Animation Module

All three interfaces have a "RUN" button that the user can click or tap to launch the graph algorithm simulation module once the graph construction step is completed. We support all combinations of directed/undirected and weighted/unweighted graphs and have implemented simulation modules for Dijkstra's shortest path and Kruskall's minimum spanning tree algorithms.

3. Evaluation

To find out how each of the three interfaces compare, we have conducted a user study. Among the goals of the user study were to determine which interface would best suit users who were somewhat knowledgeable in graph theory. For the user study, we recruited 10 students who had recently taken an algorithms course. We asked the subjects to construct and manipulate different types of graphs with varying complexities using all three interfaces. Out evaluation has revealed that the hybrid interface is preferred by users over only pen-based and only WIMP-based input methods. We believe the main reason for this is the relatively fragile nature of the digit recognition. Although the recognition rates were comparable to that of the Tablet PC SDK recognizers, digit recognition had more misrecognitions compared to the node/edge recognition.

4. Related Work

Previous research in interpreting free-hand pen input can be categorized as online methods that interpret ink as it is laid on the drawing surface, and offline systems that interpret previously recorded drawings. Our system uses online recognition.

SILK and DENIM [2] are amongst the earliest tools to demonstrate the use of intelligent processing of free-hand input in enhancing user interfaces. Others systems recognize sketches from a number of domains including electrical circuit diagrams, UML diagrams, graphs and family trees (see [4] for a comprehensive list). Among these systems, some put more emphasis on the usability and user interface issues of their application, while others aim to push forward the state of the art in sketch recognition. From this perspective, our work is closer to the first group of work, because our main contribution is our user study which compares three different user interfaces of varying degrees of intelligent ink processing to achieve the same task. Rather than focusing on developing the best recognition techniques, we have explored the trade-off between the added convenience of doing intelligent ink-processing and the inconvenience created by misrecognitions. We believe we have identified a mixture of pen-based and WIMP-based input methods that significantly outperform either method alone.

5. Discussion and Future Work

We presented a pen-based interface for constructing weighted and unweighted graphs which functions as a front-end to a shortest path algorithm simulator that we have developed. We compared the usability of this pen-based interface to that of a WIMP-based interface and a hybrid interface that uses a mixture of pen-based and WIMP-based input methods. Our evaluation revealed that despite the use of standard digit recognition algorithms, relatively high misrecognitions at this stage significantly degrades usability. It is very likely that a combination of the relatively low resolution sampling rates for pen-based input and the kinesthetic properties of writing on a capturing device as opposed to paper is responsible for the low recognition rates. This suggests that building robust user independent digit recognizers is a critical research problem despite the high recognition rates obtained for scanned images of digits.

References

[1] H. Dibeklioglu. A sketch recognizer for graphs. *Engineering Project Report, Yeditepe Univ. Dept. of Computer Eng.*, 2006.

[2] M. A. Hearst, M. D. Gross, J. A. Landay, and T. F. Stahovich. Sketching intelligent systems. *IEEE Intelligent Systems*, 13(3):10–19, May-June 1998.

[3] D. Liu and T. Chen. Soft shape context for iterative closest point registration. *IEEE International Conference on Image Processing*, 2004.

[4] T. M. Sezgin and R. Davis. Sketch interpretation using multi-scale models of temporal patterns. *IEEE Computer Graphics and Applications*, 1(27):28–37, Jan-Feb 2007.

[5] B.-J. van der Zwaag. Handwritten digit recognition: A neural network demo. *LNCS 2206*, pages 762 – 771, 2001.