Cost Evaluation of Interactively Correcting Recognized Engineering Drawings

Wenyin Liu[1], Liang Zhang[2], Long Tang[2], and Dov Dori[3]

[1] Microsoft Research, Sigma Center,
#49 Zhichun Road, Beijing 100080, PR China
wyliu@microsoft.com
[2] Department of Computer Science and Technology, Tsinghua University,
Beijing 100084, PR China
[3] Faculty of Industrial Engineering and Management,
Technion – Israel Institute of Technology, Haifa 32000, Israel
dori@ie.technion.ac.il

Abstract. We present a new scheme, which is based on the cost of interactively correcting detection errors, for performance evaluation of engineering drawings recognition algorithms. We correct an actual output graphic objects so that they to the expected output in a graphics editing tool using GUI, like Autocad. Through experiments, We define the time spent on the editing operations as the output element‘s editcost, which we use as a performance evaluation index.

1 Introduction

Performance evaluation of automatic engineering drawings vectorization has been proposed in several ways, including the number of matches (Kong et al. 1996, Hori and Doermann 1996, and Phillips et al. 1998), detection accuracy (Liu and Dori 1997, 1998), and edit cost (Phillips and Chhabra 1999). These proposed protocols provide indices for quantitative comparison among vectorization systems. However, the edit cost index (Phillips and Chhabra 1999) is more useful than others since the imperfect results of vectorization systems require manual corrections, so the parameter we wish to minimize is the edit cost. Edit cost is also based on the number of matches between the ground truths and detected entities.

Each class of graphic entities and each type of detection error may require a different amount of edit cost. For example, a short line, falling completely inside the editing window and a long one covering more than one window may require different amount of effort in the correcting action. Hence, the edit cost index based solely on the number of matches may not reflect the real edit cost required by the actual editing operations involved in correcting the detection error in terms of the time spent on the correction.

Atul K. Chhabra and D. Dori (Eds.): GREC'99, LNCS 1941, pp. 329-334, 2000.

In this paper we propose an alternative definition of the edit cost index based on the actual cost of interactively/manually correcting the imperfect detection. The graphics-editing tool we used in the experiments is Autocad R14. We have measured the edit cost for the graphic classes of points, lines, arc, circles, and polylines. The proposed scheme can be extended to other classes of graphics.

The total editcost of the entire drawing of graphic object recognized by an algorithm is the sum of editcost of all recognized graphic objects. In order to give an normalized and uniform index of editcost performance of an algorithm, an index formed by the total editcost divided by the cost of redrawing all the graphic objects (actually redrawing all the ground truths) is used compare performance of the same algorithm on different drawings. In this case, the value of the uniform index is between 0 (which is best since no cost is needed) and 1 (which is the worst and means the cost is the same as totally redrawing the drawings). The smaller the edit cost index, the better the system. We have used the defined editcost indices to evaluate the MDUS system over the test images for the second IAPR graphics recognition contest (Chhabra and Phillips 1997) and the results are presented.

2 Basic Operations of Graphics Editing and Their Editcost

The editing operations (in typical graphics editing tools, e.g., Autocad) of all classes of graphic entities may consist of the following basic operations, the edit costs of which are also defined. We assume that the raster drawing is used as the background so that the locations of expected graphic objects (ground truth) are clear and can be used as guidance. The editing of other attributes (e.g., line style and color) of graphic objects are not considered in this paper.

1). Mouse click for picking an object (e.g., a point, a graphic object, etc., the picked object usually is distinguished from others) or click a window object (e.g., a button, scrollbar, etc.) with a cost of

$$C_{po} = a \tag{1}$$

where, a is a constant that can be determined from experiments.

2). Locating a point (e.g., for redrawing an endpoint of a new line or for precisely drop a moving point when correcting a line) with a cost of

$$C_{lp} = b \tag{2}$$

where, b is a constant that can be determined from experiments.

3). Drag and drop for moving a point or graphic object to a new position with a cost of a linear approximation:

$$C_d = k_1 * d_{sd} + c \tag{3}$$

where d_{sd} is dragging distance from a start point to an end point, k_1 is a constant which is equivalent to the dragging speed, c is a constant. Actually, this operation includes two steps. The first step is dragging the object by a distance and the second step is precisely dropping it to a new location. The cost second step is exactly the same of Eq. (2). Therefore there should be a relationship be b and c: $c > b$.

4). Searching the next point for correction, C_{sp}. This operation is an important step during graphics editing. For instance, if the two endpoints of a line are not inside the

same window, after editing the first endpoint, finding the second endpoint requires some time, which is denoted by C_{sp}. The definition of C_{sp} is complex for many situations. However, we can approximate it using the cost of scrolling the window from the current point located at (x_1, y_1) to the expected point located at (x_2, y_2):

$$C_{sp} = a((x_2 - x_1)/w + (y_2 - y_1)/h) \quad (4)$$

where w is the width (the default value is 640 pixels) and h is the height (the default value is 480) of the editing window.

We have used Autocad R14 as a test bed to evaluate the real editcost of some typical graphics objects. The user is a student familiar with PC and Windows though not an Autocad professional. His experience may be regarded as of typical users.

In the experiment the user tests picking up 10 points, 10 lines, 10 circles, and 10 arcs in the current screen are. The average of picking up an object is 1.19 seconds. Hence, the constant in Eq. (1) is

a=1.19

The experiments on locating (or dropping) a point yield a result of:

b=3.03

The reason that $b > a$ is that picking up an object do not need precise location while locating or dropping an object to a precise location need more deliberation.

The constants in Eq. (3) are also obtained by several dragging experiments. Their average values are:

k_1=0.0083; c=3.80

3 Costs of Correcting and Redrawing Typical Graphic Objects

Having defined the editcost of those basic operations, the correcting cost or redrawing cost of a graphic object is defined as

$$Ec = \sum_{i=1}^{n} C_i \quad (5)$$

where C_i (i=1..n) is the editcost of the i^{th} step operation involved in the correction or redrawing of the graphic entity. The typical classes of graphic objects may include the following basic operations.

1) Point

The editing of a point consists of two steps: picking up the point and dragging and dropping it to the destination. Therefore the cost consist of Eqs. (1) and (2):

$$Ec = C_{po} + C_d \quad (6)$$

2) Line

Usually we have two ways of correcting a line. One way is move the line such that one of its endpoints coincide with the expected one and then dragging the other endpoint to its correct location, as shown in **Figure 1**. Another way is correcting the two endpoints separately, as shown in **Figure 2**. The first method is cost effective if the line only translated from its expected location and other attributes are the same, in which case, only one step of moving (of the entire line) is enough. For other cases, the moving of both the two endpoints are required. Especially when the line is quite long

and the two endpoints are not shown simultaneously in the same screen area, the second way is recommended, whose cost includes moving of two endpoints (Eq. (5)) and the searching for the next point for correction (Eq. (3)):

$$Ec = \delta_1(C_{po} + C_{d1}) + \delta_2(C_{po} + C_{d2}) + C_{sp} \tag{7}$$

where δ_i (i=1 or 2) is 0 if the ith is at its expected location and needless to be corrected, and is 1 otherwise.

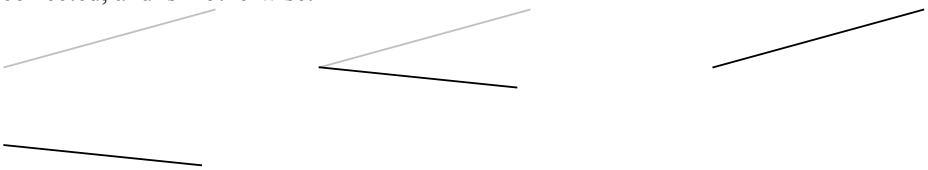

Figure 1. The first way of correcting a line. (a) original state, (b) move the line to coincide one endpoint, and (c) coincide another endpoint

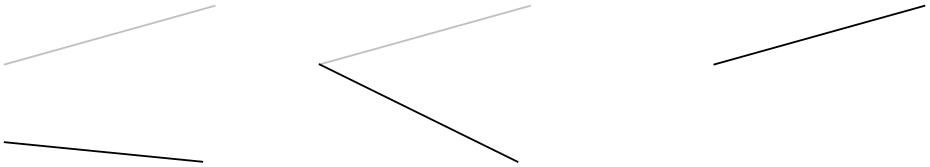

Figure 2. The second way of correcting a line. (a) original state, (b) coincide one endpoint, and (c) coincide another endpoint.

While redrawing a line consists of precisely locating the two endpoints and finding the second endpoint from the first endpoint. Therefore the cost of redrawing a line is

$$Rc = 2C_{lp} + C_{sp} \tag{8}$$

3) Circle

We correct a circle by correcting the center and the radius of the circle (actually a point on the circle). The correcting cost and is equivalent to that of correcting a line in Eq. (5) and the redrawing cost is equivalent to that of redrawing a line.

4) Arc

Similarly to correcting a line, we correct an arc by correcting the three points (two endpoints and the center or another point on the arc). The correcting cost is

$$Ec = C_{po} + \delta_1 \cdot C_{d1} + C_{sp-1'2} + \delta_2 \cdot C_{d2} + C_{sp-2'3} + \delta_3 \cdot C_{d3} \tag{9}$$

The redrawing cost of an arc includes 3 times of locating points in Eq. (2) and twice finding the next point in Eq. (4).

5) Polyline

We correct a polyline by correcting each point respectively. The correcting cost is

$$Ec = C_{po} + \sum_{i=1}^{n} \delta_i \cdot C_{di} + \sum_{j=1}^{n-1} C_{sp-j'j+1} \tag{10}$$

The redrawing of a polyline includes locating the points and finding the next points sequentially.

After defining the editcost and redraw cost of graphic objects, we observed that the calculated editcost may be greater than the redraw cost in some cases, e.g., a short recognized line matched with a long ground truth. Particularly in these cases we use the redraw cost as the final editcost of the object.

The editcost of the entire drawing is defined as

$$Ec_{total} = \sum_{j=1}^{m} Ec_j \tag{11}$$

where j=1..m are the enumeration of the ground truths of the drawings. Note that some ground truth objects may not have a corresponding detection. The correction for this kind of error is simply redrawing the entire object, whose editcost is the redrawing cost.

Although the total edit cost of the drawing, defined in Equation (10), can be used as a performance evaluation index of the vectorization, it is not normalized and hence it cannot be used to compare among different drawings. In order to get a comparable edit cost index among all drawings, the edit cost index is normalized by dividing by the total redrawing cost for all the ground truths. In this case the normalized editcost falls in the range 0 to 1, it provides a comparable index not only among algorithms/systems but also among drawings and can serve as the unique index to indicate the relative performance evaluation of each vectorization system. The smaller the normalized editcost index, the better the system.

4 Cost Evaluation of MDUS

We have used the editcost definition to evaluate the Machine Drawing Understanding System (MDUS) (Liu and Dori 1996). Two test images (ds08.tif and ds33.tif) downloaded from the web page developed by Chhabra and Phillips (1997) for the second IAPR graphics recognition contest. We have only evaluated the editcost of lines and arcs and show the result in **Table 1**. Editcost of text is not evaluated since MDUS's ability on text segmentation is not so strong and OCR is not included.

The editcost is evaluated at different position tolerance level: 1, 3, and 5 pixels. The points within the allowed tolerance are not corrected and the corresponding editcost are saved. From **Table 1** we can see that the relative editcost of the two recognized drawings are very high. The editcost is still about two thirds of the cost of totally redrawing even though the tolerance is 5 pixels. Probably this is the reason why the users are unsatisfied with the current raster to vector conversion products.

Table 1. Editcost of lines and arcs recognized by MDUS on two test images

Image	Redraw Cost (seconds)	Edit Cost (seconds) at Different Tolerance (pixels)			Normalized Editcost Index at Difference Tolerance (pixels)		
		1	3	5	1	3	5
Ds33	2311	1945	1691	1562	0.84	0.73	0.67
Ds08	2096	1572	1315	1296	0.75	0.63	0.62

5 Summary

We have defined the editcost for some classes of graphics within the Autocad GUI environment and get the value of them only through limited experiments. However, the editcost within other editing tools, or by different people using different editor options and at different skill levels may have different evaluation of editcost. The purpose of this paper is provide a scheme of evaluating the editcost of graphic recognition results. More experiments should be done to obtain more objective evaluation of the editcost of graphic objects.

A conclusion may be drawn from the experiments that the relative editcost of recognized drawings is very high and this may be the reason why the current users are reluctant to use the raster to vector conversion products. We researchers may have to make some changes to the state of the art.

References

1. Chhabra A and Phillips I (1997) Web pages for the Second International Graphics Recognition Contest—Raster to Vector Conversion. http://graphics.basit.com/iapr-tc10/contest.html
2. Chhabra A and Phillips I (1998) The Second International Graphics Recognition Contest—Raster to Vecter Conversion: A Report. In: Tombre K, Chhabra A (eds). Graphics Recognition—Algorithms and Systems (Lecture Notes in Computer Science, Vol. 1389). Springer, 1998, pp390-410.
3. Hori O and Doermann DS (1996) Quantitative Measurement of the Performance of Raster-to-Vector Conversion Algorithms. In: Kasturi R, Tombre K (eds) Graphics Recognition -- Methods and Applications (Lecture Notes in Computer Science, vol. 1072). Springer, Berlin, pp 57-68
4. Kong B, Phillips IT, Haralick RM, Prasad A, Kasturi R (1996) A Benchmark: Performance Evaluation of Dashed-Line Detection Algorithms. In: Kasturi R, Tombre K (eds) Graphics Recognition -- Methods and Applications (Lecture Notes in Computer Science, vol. 1072). Springer, Berlin, pp 270-285
5. Liu W and Dori D (1996) Automated CAD Conversion with the Machine Drawing Understanding System. In: Proc. of 2nd IAPR Workshop on Document Analysis Systems, Malvern, PA, USA, October, 1996, pp 241-259
6. Liu W and Dori D (1997) A Protocol for Performance Evaluation of Line Detection Algorithms. Machine Vision Applications 9(5):240-250
7. Liu W and Dori D (1998) Performance Evaluation of Graphics/Text Separation. In: Tombre K, Chhabra A (eds). Graphics Recognition -- Algorithms and Systems (Lecture Notes in Computer Science, Vol. 1389). Springer, 1998, pp 359-371.
8. Phillips IT, Liang J, Chhabra A, and Haralick RM (1998) A Performance Evaluation Protocol for Graphics Recognition Systems. In: Tombre K, Chhabra A (eds). Graphics Recognition -- Algorithms and Systems (Lecture Notes in Computer Science, Vol. 1389). Springer, 1998, pp 372-389
9. Phillips I and Chhabra A (1999) Empirical Performance Evaluation of Graphics Recognition Systems. IEEE Trans. on PAMI, 21, 9, pp 849-870