Empirical Performance Evaluation of Graphics Recognition Systems

Ihsin T. Phillips, *Member*, *IEEE*, and Atul K. Chhabra, *Senior Member*, *IEEE*

Abstract—This paper presents a methodology for evaluating graphics recognition systems operating on images that contain straight lines, circles, circular arcs, and text blocks. It enables an empirical comparison of vectorization software packages and uses practical performance evaluation methods that can be applied to complete vectorization systems. The methodology includes a set of matching criteria for pairs of graphical entities, a set of performance evaluation metrics, and a benchmark for the evaluation of graphics recognition systems. The benchmark was tested on three systems. The results are reported and analyzed in this paper.

Index Terms—Empirical evaluation, benchmark, graphics recognition, engineering-drawing.

1 Introduction

Systems which convert existing paper-based drawings into electronic format are in great demand. Several such systems have been developed in the last few years, both as commercial systems and research prototypes. The performance of these systems is either unknown or only reported in a limited way by the system developers. An evaluation of these systems, or their subsystems, would contribute significantly to the advancement of the field. This has motivated some recent work on performance evaluation [1], [2], [3], [4] and two international graphics recognition contests [5], [6].

In this paper, we present an empirical methodology to evaluate the performance of graphics recognition systems. The methodology is designed to work with images that contain solid or dashed straight lines, circles, and circular arcs, as well as text regions. The graphical entities are limited to the above types. Nevertheless, the evaluation methodology is quite useful since all engineering drawings use mostly a combination of these geometric elements. This methodology differs from previous work in two ways. First, it can evaluate a complete raster to vector conversion system, not just a single module, such as dashed line detection. Second, the performance metrics are goal-directed, i.e., they are motivated by the ultimate goal of the conversion of a given drawing, which often involves some manual effort to correct the mistakes made by the system.

The methodology includes a set of matching criteria for pairs of graphical entities, a set of performance evaluation metrics, and a benchmark for the evaluation of graphics recognition systems. The methodology provides an empirical comparison of vectorization software packages. It uses practical performance evaluation methods that can be applied to complete vectorization systems. The methodology has been implemented and tested on three large systems [7], [8], [9]. The performance evaluation of these three systems is presented in this paper. We hope that the methodology we present here will help assess the state of the art in graphics recognition and highlight the strengths and weaknesses of the current vectorization technology and the evaluation methods.

This paper is organized as follows: In Section 2, we give a brief review of some related work. In Section 3, we describe the parameters for the graphical entities in the context of our evaluation. In Section 4, we give an overview of the performance evaluation. In Section 5, we describe the computation and processing of the match score table. In Section 6, the matching algorithms and criteria for pairs of entities are described. The performance metrics are given in Section 7. The benchmark specifications are given in Section 8 and the results of performance evaluation of the three tested systems are given in Section 9. We conclude the paper with a discussion on possible future work (Section 10).

2 Previous Work

Performance evaluation and benchmarking have been gaining acceptance in all areas of computer vision. An overview of this area is available in [10]. Performance evaluation of graphics recognition is still a very young field; objective and quantitative methods for evaluation of graphics recognition have been proposed very recently [1], [2], [3], [4]. Kong et al. [1] propose a quantitative method for evaluating the recognition of dashed lines. Hori and Doermann [2] propose a quantitative performance measurement methodology for task-specific raster to vector conversion. Wenyin and Dori [3] present a protocol for evaluating the recognition of straight and circular lines. All of these methods are limited in their applicability.

Kong et al. [1] use angle, distance, relative overlap, and offset between line segments for evaluating line matches

- *I.T. Phillips is with the Department of Computer Science/Software Engineering, Seattle University, Seattle, WA 98122. E-mail: yun@seattleu.edu.*
- *A.K. Chhabra is with Bell Atlantic Network Systems, Advanced Technology, 500 Westchester Ave., White Plains, NY 10604. E-mail: atul@basit.com.*

Manuscript received 3 Mar. 1999.
Recommended for acceptance by D. Dori.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 109362.

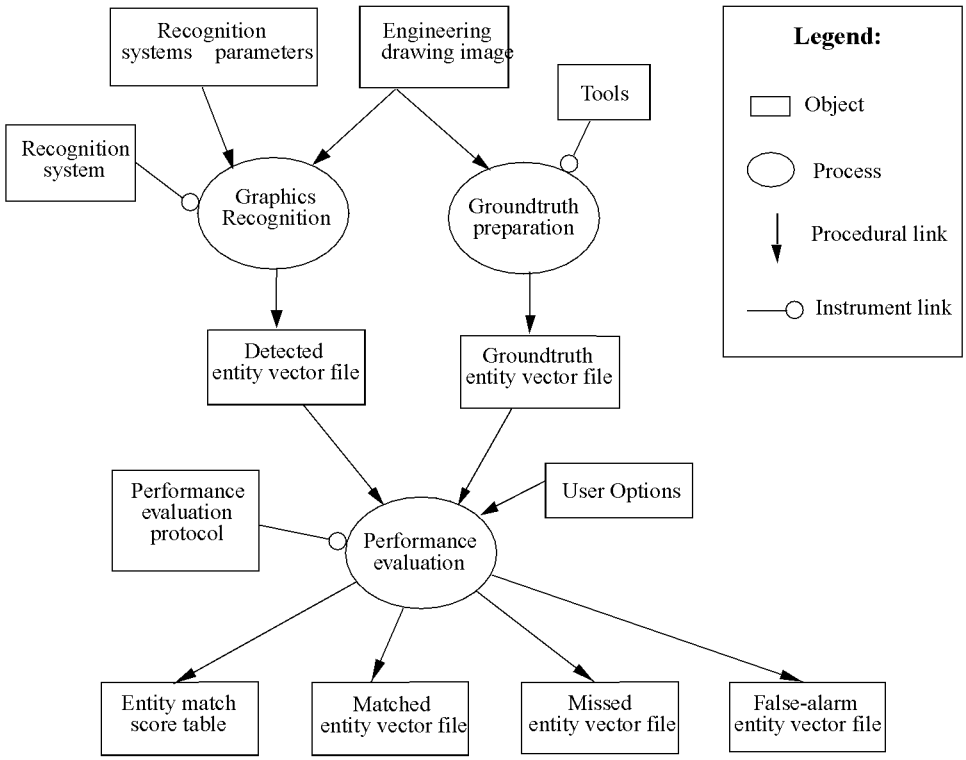

Fig. 1. The object-process diagram of our evaluator.

and for detecting line styles. They use several arbitrary and rigid thresholds. They do not allow for fragmentation of detected lines.

Hori and Doermann [2] instantiate and extend Haralick's framework for performance characterization in image analysis [11], in an application-dependent manner, for measuring the performance of raster to vector conversion algorithms. The "applications" addressed in the work are thinning, medial line finding, and line fitting—all low level techniques that do not completely constitute vectorization. It is hard to extend the work to evaluate a complete vectorization system. Hori and Doermann's protocol does not distinguish between detection rate and false alarm rate. It does not include an overall evaluation metric. It does not allow for fragmentation of detected lines.

Wenyin and Dori [3] propose performance evaluation indices for straight and circular line detection. Detection and false alarm rates are defined at both the pixel level and the vector level. Use of pixel level performance indices (measures of shape preservation) is not appropriate for real images that contain severe distortion introduced by warping and other defects in the hard copy drawing and by the scanning/imaging system. On such images, attempts to obtain a high pixel recovery index would unnecessarily require the detected vectors to be true to the distorted shape of the imaged lines, thereby making the detected lines fragmented. For such images, the pixel recovery index needs to be assigned less weight than the vector recovery index. However, there is no way to predetermine the right relative weights for the pixel and vector recovery indices.

The work of Wenyin and Dori [3] emphasizes that correct recognition of highly complex objects (that are harder to detect correctly) should carry more weight than recognition of objects that are simple. The performance evaluation methodology that we present in this paper approaches the task of evaluation from the opposite angle. We don't look at the complexity of the entities to be recognized. Instead, in our view, the true measure of performance has to be goal directed. The goal of line drawing recognition is to convert a paper copy or a raster image of a line drawing into a useful form (such as a vector CAD file). How well a graphics recognition system works should be measured by how much manual effort is required to correct the mistakes made by the system, not by how well it recognizes difficult shapes.

Match-Score Table

	g1	g2	g3	g4	g5	g6	g7	g8	g9	g10
d1							.85		.14	
d2				1.0						
d3			.1			.9		.1		
d4					.95		.9			
d5	.25	.3	.86					.3	.88	
d6		1.0								
d7		.06	.91						.93	
d8		.91								

Fig. 2. An example of a match-score table.

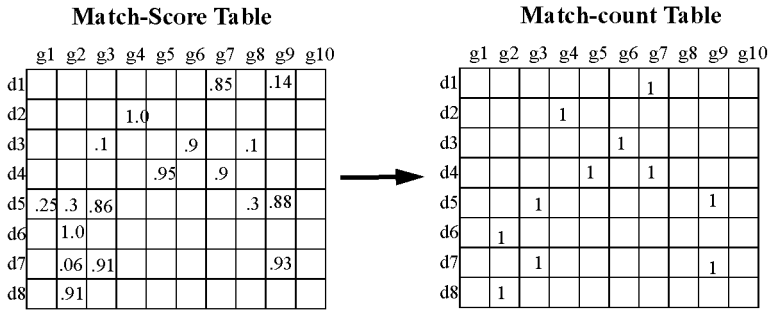

Fig. 3. An example of a match-count table (on the right).

Wenyin and Dori weight all true positives and false positives by their respective lengths. This is inappropriate if the goal of the evaluation is to measure the cost of postprocessing operations that are necessary to correct the mistakes of vectorization. Although we have not verified this empirically, we feel that the time for manual postprocessing does not depend significantly on the length of a true positive, a missed entity, or a false positive. Time taken for adding or deleting a line in a CAD tool does not depend significantly on the length of the line. Many CAD tools have user interfaces that make it just as easy to place the end points of a long line as those of a short line. They accomplish this by automatically bringing up a pop-up window showing a zoomed in view of the surroundings of the cursor.

Neither of the above methods addresses the extraction or separation of text from graphics. It is not possible to evaluate graphics recognition systems on realistic drawings without accounting for text in the drawings. Wenyin and Dori [4] propose a protocol for evaluating text-graphics separation. In this protocol, the quality of the recognized text boxes is measured using Q_b, the basic quality, and Q_{fr}, the fragmentation quality. The protocol does not explicitly penalize overlapping text boxes; they are penalized in an indirect way. Q_{fr} implicitly penalizes overlap among recognized text boxes. For N recognized text boxes that are identical and have a 100 percent overlap with a text box in the ground-truth, Q_{fr} would be $1/\sqrt{N}$. The theoretical basis for penalizing overlapping text box recognition by $1/\sqrt{N}$ is not stated. Moreover, the protocol of [4] does not allow one to accept one of the N identical text boxes as a good match and to label others as false alarms. Therefore, this penalty term cannot be used to measure postprocessing/editing cost.

3 Definitions of Graphical Entities

The graphical entities within the domain of our evaluation are of seven types: solid and dashed lines, solid and dashed arcs, solid and dashed circles, and text areas. Specification of the parameters for these seven types are given below. In the following, the units of linear dimensions are pixels and the units of angles are degrees measured clockwise from the x axis. The x axis points rightward and the y axis points downward.

Solid or dashed line: For a solid or dashed line segment, the parameters are the entity type indicator (a solid line or a dashed line) and the x and y coordinates of the two end points. No special ordering of the end points is required here.

Solid or dashed circle: For a solid or dashed circle, the parameters are the entity type indicator, the x and y coordinates of the center, and the radius.

Solid or dashed arc (partial circle): The parameters are the entity type indicator (a solid or a dashed arc), the x and y coordinates of the center, the radius, and the beginning and the ending angles of the arc in degrees. The beginning and ending angles are ordered so that the arc is drawn clockwise.

Text area: A text area is represented by the rectangular bounding box of the text and the orientation of the baseline of the text. The parameters are the entity type indicator (a text area), the x and y coordinates of any two opposite corners of the bounding box, and the orientation of the text baseline. The orientation is only used to provide a constraint to construct the box. It does not matter if the orientation is off by a multiple of 90°.

4 Performance Evaluation

We measure the performance (accuracy) of a detection algorithm by counting the number of matches between the entities detected by the algorithm and the entities in the ground-truth, and the numbers of misses and false alarms. The counting scheme we use here does not take into

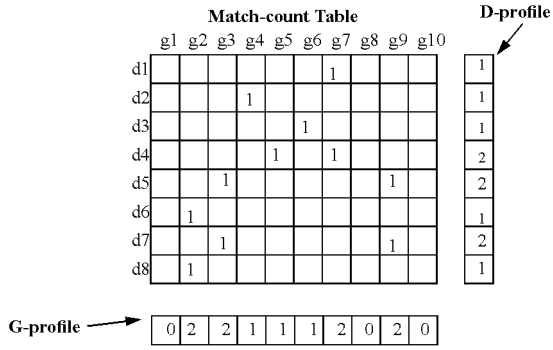

Fig. 4. The two projection profiles for the match-count table of Fig. 3.

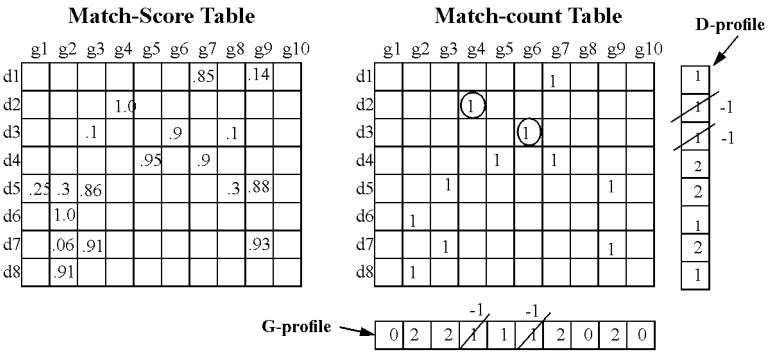

Fig. 5. One-to-one matching (the resulting one-to-one matches are circled).

account the varying complexity for detecting different types of entities. As stated earlier, in our view, goal directed performance evaluation requires that you weight the cost of correcting a mistake in recognition by how expensive it is to correct the mistake. This is not necessarily the same as assigning weights to the objects to be recognized based on the complexity of the objects. We feel it would be logical to assign some kind of goal directed weights to the entities. These weights should be empirically determined. For the current work, however, we assign equal weights to the types of entities we consider in this evaluation.

The present evaluation methodology accounts for the quality of matches between the ground-truth and the detected entities using the match scores. We set an acceptance threshold and a rejection threshold on the match scores. When a pair of a ground-truth entity and a detected entity have a match score higher than the acceptance threshold, the match is accepted as a one-to-one match. When the match score is below the rejection threshold, the match is rejected. When the score is in between, partial matching is considered. A recognition system is considered to be perfect if each and every one of the entities in the detected list matches one and only one entity of the same type in the ground-truth list and vice versa. The process of counting matches is described in more detail in the rest of this section.

A recognition system is tested on a set of preselected test images for which ground-truth vector data is available. For each test image, the recognition result produced by the system is matched by the performance evaluator against the corresponding ground-truth. This produces the counts of one-to-one matches, one-to-many matches, many-to-one matches, and false-alarms and misses. Fig. 1 shows an object-process diagram [12], [13] of this evaluator.

5 The Match Score Matrix

The matching procedure is as follows: First, entities within the recognition result file are grouped and indexed by their type (as discussed in Section 3, there are seven entity types). The ground-truth entities are also grouped and indexed in the same fashion. Next, pairwise match strengths (or match scores) are computed for each recognized entity paired with each permissible ground-truth entity. Match scores range from 0 to 1, 1 being a perfect match. The computed scores are stored in a match score table (or matrix). Fig. 2 illustrates such a table. In this table, read blanks as zeros.

Note that entries in each row i of the match-score table represent matching results from the ith entity in the recognition result (the D-list, or the list of detected entities) to all permissible entities in the ground-truth (the G-list, or the list of ground-truth entities). We attempt to match a given entity only with certain types of entities. For example, text blocks are only matched against text blocks. Also, at present, solid lines are only matched with solid lines or arcs, not with dashed ones. Within a given row, a single entry with a high match score indicates that the pair corresponding to that entry has a good match.

The performance evaluator searches within the table for pairs of one-to-one matches. We call a pair a one-to-one match if the matching score for this pair is equal to or above the evaluator's acceptance threshold (the choice of this threshold parameter will be discussed later).

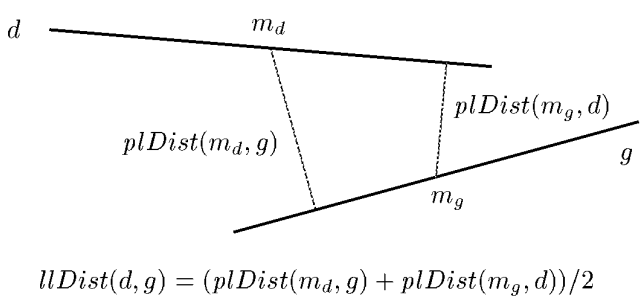

Fig. 6. Line-to-line distance of two line segments.

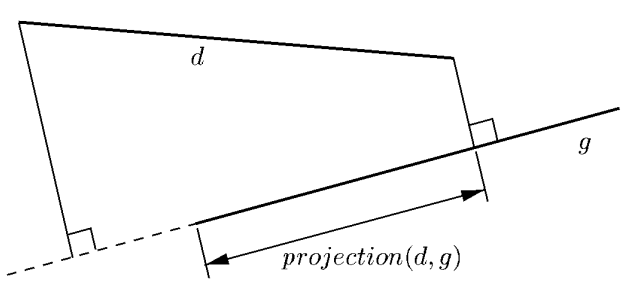

Fig. 7. Projection of the detected line d onto the ground-truth line g.

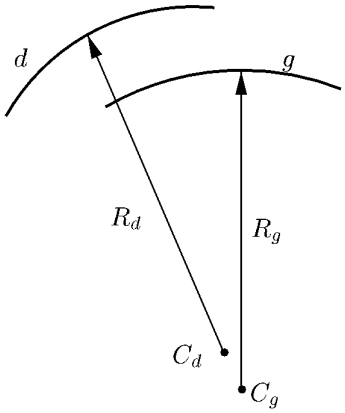

Fig. 8. An arc pair: a detected arc d and a ground-truth arc g.

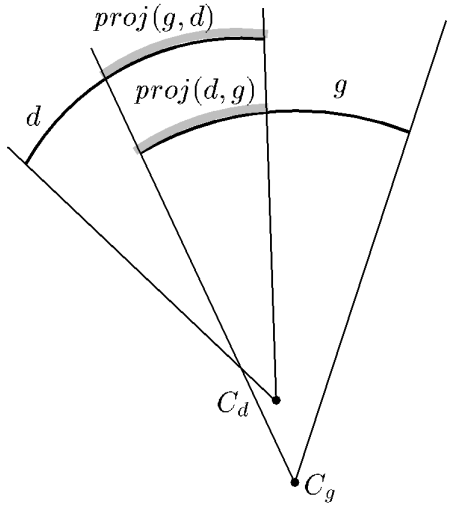

Fig. 9. The projection of arc d onto arc g and that of g onto d.

Once a pair is labeled as a one-to-one match, it is excluded from the search space. Then, the evaluator searches for one-to-many matches and many-to-one matches.

A g_one-to-many match is a ground-truth entity that "partially" matches with two or more entities in the detected result (i.e., two or more d_many-to-one matches). A pair is called a partial match if the pair's match score is above the rejection threshold (0.1) but is below the acceptance threshold. For example, when a line cuts through a circle, a recognition system may detect the line as three connected collinear lines. The ground-truth line thus produces several partial matches with detected lines. In this situation, we have a single g_one-to-many match and three d_many-to-one matches. Similarly, some recognition systems may recognize the circle as two connected arcs. Here, the ground-truth circle has partial matches with the two detected arcs. This would be a single g_one-to-many match and two d_many-to-one matches.

Likewise, a d_one-to-many match is a detected entity that "partially" matches two or more entities in the ground-truth. For example, a group of short collinear lines in the ground-truth may be recognized as a single dashed line. In this case, many ground-truth lines would be matched with one detected line. This would count as a single d_one-to-many match and several g_many-to-one matches.

False-alarms (erroneous detections) are those entities in the result file that do not match with any entity in the ground-truth (i.e., their match score with any entity in the ground-truth file is less than the rejection threshold). Misses are those entities in the ground-truth that do not match with any entity in the result file.

The outputs of this matching procedure are the counts of one-to-one matches, one-to-many matches, many-to-one matches, false-alarms, and misses.

A complete description of this matching procedure follows.

5.1 Computing One-to-One Matches

The protocol for computing one-to-one matches is as follows:

Step 1. We compute a two-dimensional match-count table from the computed match-score table. The entry *MatchCount* (i,j) is set to 1 if the *MatchScore*(i,j) is greater or equal to *acceptance threshold*; otherwise, it is set to zero. For the purpose of illustration in this section, we use an acceptance threshold of 0.85. In reality, the acceptance threshold is variable. In fact, later, in the results section, we plot the performance metric versus the acceptance threshold.

Fig. 3 illustrates the match-count table (on the right) which is computed from the match-score table on the left.

Step 2. Two projection profiles, D-profile and G-profile, are computed from the match-count table, the result of Step 1.

The entry $D(i)$ is computed as the sum of the matches in the ith row of the match-count table. Likewise, the entry $G(j)$ is computed as the sum of the matches in the jth column of the match-count table. Fig. 4 illustrates the D-profile and the G-profile computed from the match-count table.

The interpretation of these two profiles is as follows:

- One-to-one matches: An ith D entity has a one-to-one match with a jth G entity, if
  - $D(i)$ is a one,
  - the *MatchCount*(i,j) is one, and
  - $G(j)$ is one.

  (If the detection algorithm produces a perfect result, all entries in the D-profile and the G-profile will be one.)
- d_many-to-one conflicts: There is a many-to-one (detected to ground-truth) conflict if an entry in the G-profile, say $G(j)$, is greater than one. That is, there are multiple D entities matching with the jth entity is G-list.
- d_one-to-many conflicts: There is a one-to-many (detected to ground-truth) conflict if an entry in D-profile, say $D(i)$, is greater than one. That is, the ith D entity matches two or more entities in the G-list.

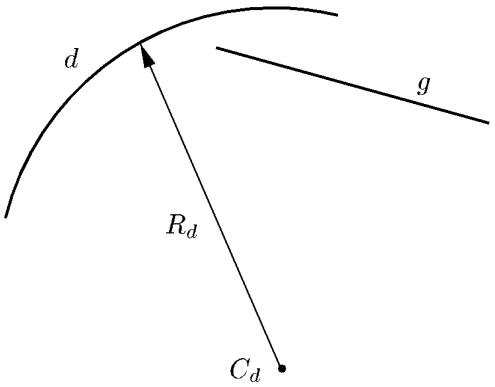

Fig. 10. An arc-line entity pair.

- False-alarms: A zero entry in the D-profile indicates that no strong match is found from this D entity to any of the entities in the G-list.
- Misses: A zero entry in the G-profile indicates that no strong match is found from this G entity to any of the entities in the D-list.

Step 3. Compute the one-to-one match list.

For each $D(i)$ that is equal to one, we attempt to locate the pair (i, j) such that both $G(j)$ and *MatchCount* (i, j) are one, a one-to-one match (d2 and d3 in Fig. 5). We put the pair (i, j) in the one-to-one match list, and set $D(i)$ and $G(j)$ to -1 (block the two entities from further consideration). Fig. 5 illustrates the result of one-to-one matching. There are two such pairs in the figure.

Step 4. Resolving d_many-to-one conflicts.

For each $D(i)$ that is equal to one (but did not produce a one-to-one match in Step 3), we locate the pair (i, j) such that *MatchCount*(i, j) is one, but $G(j)$ is greater than one. There are three such pairs in Fig. 5: (d1, g7), (d6, g2), and (d8, g2).

Without loss of generality, let $D(i)$ and $D(k)$ be one and let $G(j)$ be two (meaning that there are two D entities, say ith and kth, matching with the jth entity in G-list, such as (d6, g2), and (d8, g2) in Fig. 5).

Case 1.

We select the pair (i, j) if *MatchScore*(i, j) >= *MatchScore*(k, j)) and $D(i)$ is one. And

- We put the pair (i, j) in the one-to-one match list;
- We set $D(i)$ and $G(j)$ to -1, and
- We decrease $D(k)$ by one.

For example, if $i = 6$, we would select the pair (d6, g2) over (d8, g2) in Fig. 5.

Case 2.

We select the pair (k, j) if *MatchScore*(k, j) > *MatchScore*(i, j) and $D(k)$ is a one. And,

- We put the pair (k, j) in the one-to-one match list;
- We set $D(k)$ and $G(j)$ to -1, and
- We decrease $D(i)$ by one.

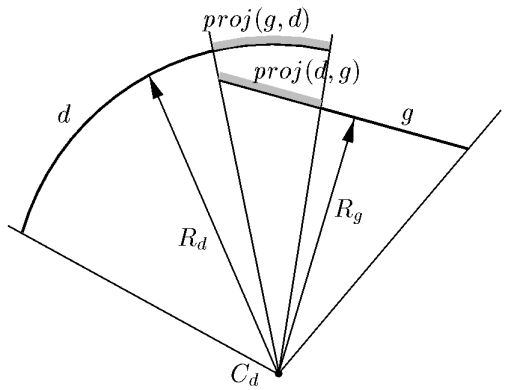

Fig. 11. Projection of arc d onto line g and that of g onto d.

For example, if $i = 8$, we still select the pair (d6, g2) over (d8, g2) in Fig. 5.

Now, we lift the restriction that, for the match pairs under consideration, $D(k)$ be equal to one. This leads to the following case:

Case 3.

MatchScore(i, j) < *MatchScore*(k, j) and $D(k)$ is greater than one, (d1 and d4 with g7, in Fig. 5.

In this case, we would not select the pair (k, j) if there is a column t, in row k such that *MatchScore*(k, t) > *MatchScore*(k, j). In this case, we would select the pair (i, j) instead. And, we handle this case as in Case 1.

In the example in Fig. 5, d1 and d4 match g7, a d_many-to-one conflict. The pair (d1, g7) is selected instead of (d4, g7) since the *MatchScore*(d4, g5) has a higher score than the pair (d4, g7).

This step is repeated until no more D(i) is equal to one.

Step 5. Resolving d_one-to-many conflicts.

For each $D(i)$ that is a two, let j and k be the two entities in G-list that match with the ith entity in the D-list. If *MatchScore*(i, j) >= *MatchScore*(i, k), we put the pair (i, j) in the one-to-one match list and set $D(i)$ and $G(j)$ to -1, and decrease $G(k)$ by one. Otherwise, we put the pair (i, k) in the one-to-one match list and set $D(i)$ and $G(k)$ to -1, and decrease $G(j)$ by one. A similar treatment is done if $G(j)$ is

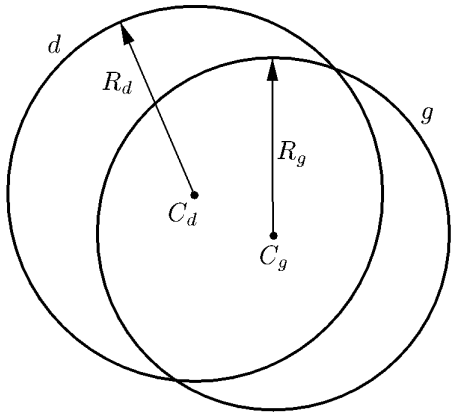

Fig. 12. A circle-circle entity pair.

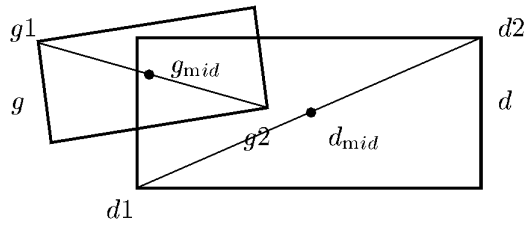

Fig. 13. A text-text entity pair.

greater than two. This step is repeated until no more $D(i)$ is two or greater.

5.2 Computing Partial Matches: One-to-Many and Many-to-One

At the end of the one-to-one entity matching (Secttion 5.1), the value of each $D(i)$ indicates whether the ith entity has a one-to-one match. In particular, if a $D(i)$ is a -1, it indicates that the ith entity in the D-file has a one-to-one match; otherwise, it indicates that it does not have a one-to-one match. The same applies for the entities in G-file. One could, for example, consider each $D(i) >= 0$ a false alarm and each $G(j) >= 0$ a missed detection.

However, it may be the case that a detected entity which does not have a one-to-one match may, in fact, match with a group of two or more ground-truth entities. For example, a detection algorithm may have located a text bounding box on the input image that includes several text lines, while those text lines are given one text bounding box each in the ground-truth file.

Therefore, to give partial credit to the detection algorithms for finding one-to-many partial matches, we do as follows:

For each $D(i) >= 0$ in the D-profile, we collect a list of all entities j in G-profile, such that $G(j)$ is also $>= 0$ (also did not have a match with any D entity) and *MatchScore*(i, j) is greater than the *rejection threshold* (so that we would not include any noise). Currently, the *rejection threshold* is set to 0.05. However, we allow users to set their own threshold. If the sum of the scores for the entities in the collected list is greater than the acceptance threshold, we consider the ith D entity having a one-to-many partial match to the j entities in the collected list. And, we set $D(i)$ and all those $G(j)$ to -1. We increment d_one-to-many by 1 and g_many-to-one by the number of ground-truth entities in the collected list.

A similar protocol is applied to find the g_one-to-many and d_many-to-one match counts.

5.3 False-Alarms and Misses

Finally, the false-alarms are those i entities having their $D(i)$ values $>= 0$, and the misses, or missed detections, are those j entities having their $G(j)$ values $>= 0$.

6 Matching Algorithms

Here, we present the techniques used for matching various types of entity pairs. First, we specify the coordinate system used in the rest of this section.

We use the column-row coordinate system, (c, r), to represent a pixel's position in an image. The origin of this coordinate system, $(0, 0)$, is at the top left corner of the image.

In the following discussion, d_i represents the ith entity in the list of detected entities and g_j represents the jth entity in the list of ground-truth entities. For simplicity, we refer to d_i and g_j as d and g, respectively.

The types of entity pairs described below are the only ones we attempt to match. For example, we attempt matching detected arcs with ground-truth lines (as described below); however, we do not attempt to match detected lines to ground-truth text regions. The procedures described below for line-to-line matching, etc., are used for matching solid-to-solid entities and dashed-to-dashed entities. We do not attempt to match dashed entities with solid entities.

6.1 Line-Line Matching

Given that d and g are both line entities, we compute their match score as follows: Here, T_a and T_d are two user defined application dependent thresholds.

Step 1: If d and g have the same endpoints (they match perfectly), we set $MatchScore(i, j)$ to 1 and skip the following steps.

Step 2: We compute the smaller angle between d and g, the two lines. If this *angle* is greater than T_a, $angle(d, g) > T_a$, we set $MatchScore(i, j)$ to zero and skip the remaining steps. Else, we continue to the next step.

Step 3: We compute the *distance*, $llDist(d, g)$, between d and g. $llDist(d, g)$ is computed as the average of the orthogonal distance from the midpoint of d to g and the orthogonal distance from the midpoint of g to d. (see Fig. 6).

If $llDist(d, g) > T_d$, we set $MatchScore(i, j)$ to zero and skip the remaining steps. Otherwise, we continue to the next step.

Step 4: Next, we compute the projection of d onto g, (d, g), (see Fig. 7). This is the overlap between g and the projection of d onto the direction g. Note that *proj* is not a commutative function.

If the $proj(d, g)$ is less than 20 percent of both d and g, we set $MatchScore(i, j)$ to zero. Else, we compute $RelativeOverlap(d, g)$ as

$$RelativeOverlap(d, g) = \frac{proj(d, g)}{\max(length(d), length(g))},$$

and set $MatchScore(i, j)$ to $RelativeOverlap(d, g)$.

6.2 Arc-Arc Matching

Let d be an arc entity in the D-entity-list and g be an arc entity in the *G-entity-list*. Let C_d and C_g be the centers of d and g and let R_d and R_g be the two radii (see Fig. 8).

We compute $MatchScore(i, j)$ (see Fig. 2), the measure of the strength of match of d and g, as follows:

Step 1: If the two arcs, d and g, are identical (with the same centers, same radii, same beginning and end angles), we set $MatchScore(i, j)$ to 1 and skip the following steps.

Step 2: We compute the point-to-point distance, $ppDist(C_d, C_g)$, between the two centers C_d and C_g. If this

TABLE 1
Counts of Various Types of Entities in the Eight Test Drawing Images

Image	Text Strings	Continuous Lines	Dashed Lines	Continuous Arcs	Dashed Arcs	Continuous Circles	Dashed Circles
ds29	82	201	31	28	0	14	12
ds30	91	250	46	24	0	19	13
ds31	135	385	19	29	0	41	17
ds32	80	319	82	13	0	16	3
ds35	186	368	0	8	0	77	0
ds36	119	801	0	105	0	3	0
ds09	141	263	13	23	0	7	0
ds07	482	791	1	0	0	86	0

distance is greater than a threshold θ_{CC}, we set $MatchScore(i, j)$ to zero, and skip the following steps.

Step 3: We compute the absolute difference between the two radii, $RadiusDiff(R_d, R_g) = |R_d - R_g|$. If this difference is greater than a threshold θ_{RR}, we set $MatchScore(i, j)$ to zero, and skip the following steps:

Step 4: We compute the ratio of these two radii,

$$RadiusRatio(R_d, R_g) = \frac{\min(length(R_d), length(R_g))}{\max(length(R_d), length(R_g))}.$$

If this ratio is smaller than a given threshold (0.85 or given by the user), we set $MatchScore(i, j)$ to zero, and skip the following steps.

Step 5: We compute the projection of d onto g, $proj(d, g)$, which is that part of the arc g that falls within the angular extent formed by the arc d and its center C_d (see Fig. 9).

Similarly, we compute the projection of g onto d, $proj(g, d)$, which is that part of the arc d that falls within the angular extent formed by the arc g and its center C_g (again, see Fig. 9). Just as in line-to-line matching, the projection function for arcs is noncommutative.

Step 6: If only one or none of $proj(d, g)$ or $proj(g, d)$ exists, we set $MatchScore(i, j)$ to zero and skip the following steps. Otherwise, we create a straight line segment between the two end points of $proj(d, g)$ and call it L_{dg}. Similarly, we create a straight line segment between the two end points of $proj(g, d)$ and call it L_{gd}.

Step 7: Next, we apply the line to line matching criteria (see Section 6.1) to the pair of lines L_{gd} and L_{dg}. The $MatchScore(i, j)$ for the pair of arcs d and g is computed as the result of the line-line match score times an adjustment factor, F. F is defined as:

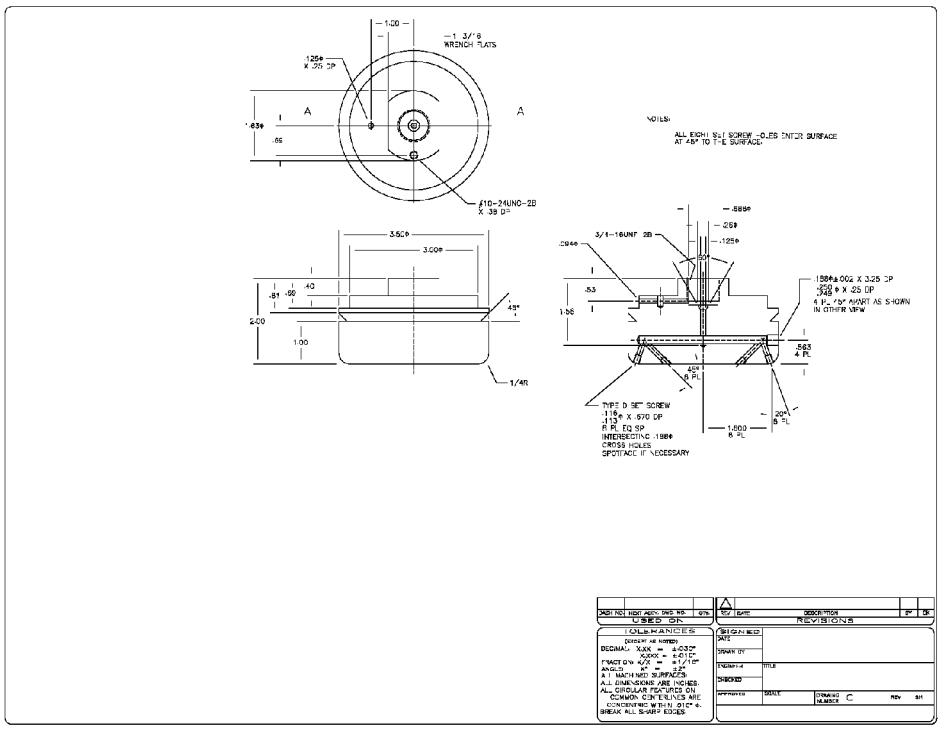

Fig. 14. Training image mech.tif (mechanical drawing).

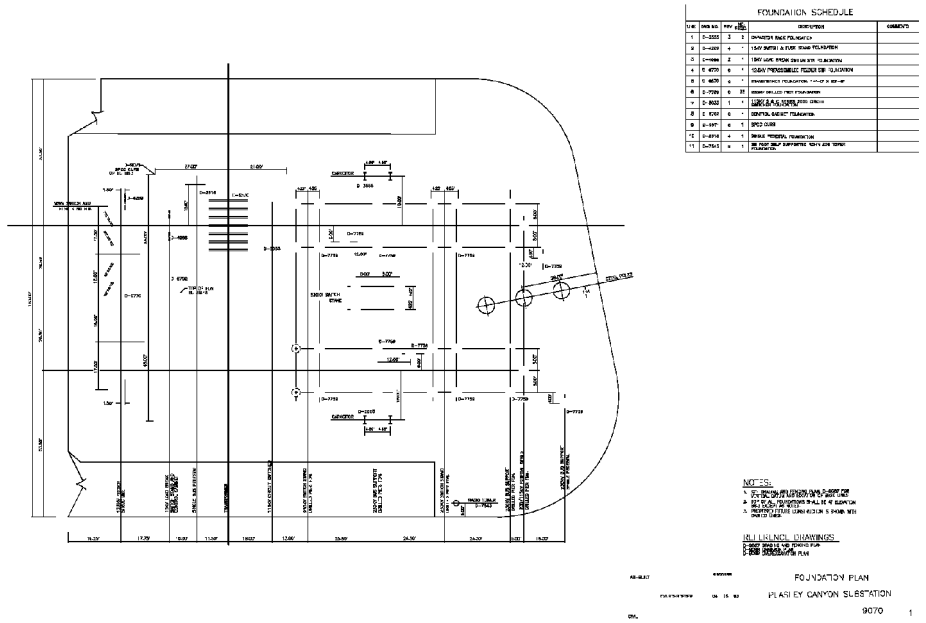

Fig. 15. Training image arch.tif (architectural drawing).

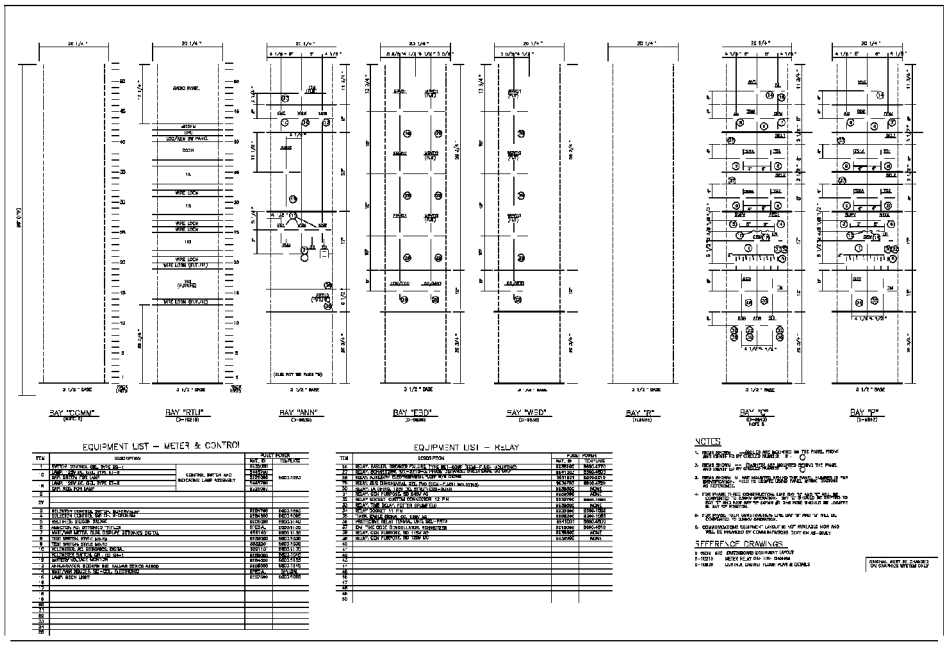

Fig. 16. Training image util1.tif (utility drawing, type 1).

$$\frac{\min(length(L_{dg}), length(L_{gd}))}{\max(length(L_d), length(L_g))},$$

where L_d and L_g are the chords of the arcs d and g, respectively. The factor F favors longer pairs of line segments over shorter line segments.

6.3 Arc-Line Matching

Let d be an arc entity in the *D-entity-list*, and let C_d be the center and R_d be the radius of d. Let g be a line entity in the *G-entity-list* (see Fig. 10).

We compute $MatchScore(i, j)$, the goodness of the match between the detected arc d and the ground-truth line g, as follows.

Step 1: We call the distance from C_d to the midpoint of g as R_g. One can think of this as constructing an artificial arc with center C_d, radius R_g, and angular extent the same as the angle subtended by the line g at the point C_d.

Step 2: We compute the absolute difference between R_d and R_g, $|R_d - R_g|$. If this difference is greater than θ_{RR}, we set $MatchScore(i, j)$ to zero, and skip the following steps:

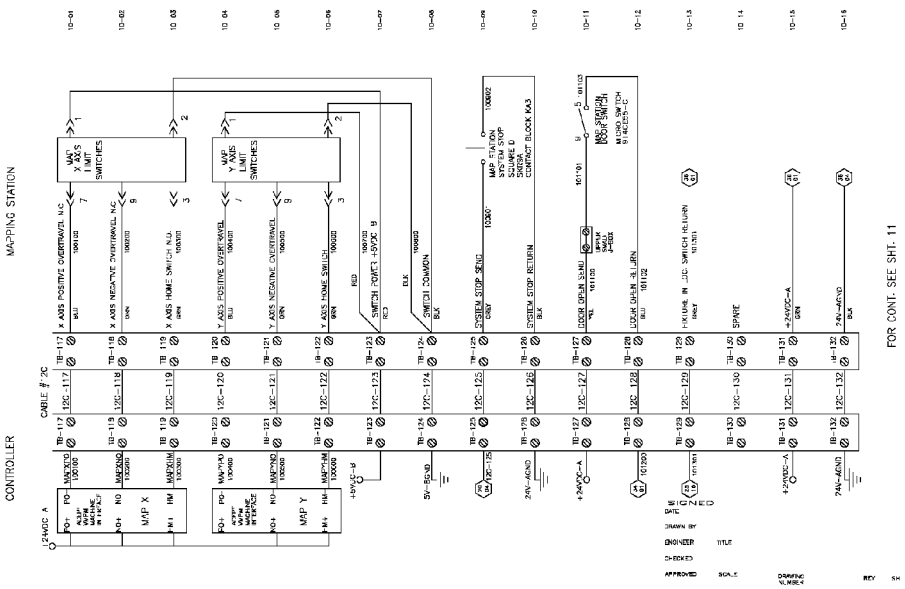

Fig. 17. Training image util2.tif (utility drawing, type 2).

Step 3: We compute the projection of arc d onto line g, $proj(d, g)$, which is that part of the line g that falls within the angular extent formed by the arc d and its center C_d (see Fig. 11).

Similarly, we compute the projection of line g onto arc d, $proj(g, d)$, which is that part of the arc d that falls within the angle subtended by the line g onto the point C_d (see Fig. 11).

Step 4: If only one or none of $proj(d, g)$ or $proj(g, d)$ exists, we set $MatchScore(i, j)$ to zero and skip the rest of the procedure. Otherwise, we create a straight line segment between the two end points of $proj(g, d)$ and call it L_{gd}. We refer to the line segment $proj(d, g)$ as L_{dg}.

Step 5: We apply the line-to-line matching method, as described in Section 6.1, to the pair of line segments L_{gd} and L_{dg}. The $MatchScore(i, j)$ for the arc d and line g is computed as the result of the line-line match score times an adjustment factor, F. F is defined as:

$$\frac{\min(length(L_{dg}), length(L_{gd}))}{\max(length(L_d), length(g))},$$

where L_d is the chord of the arc d.

In case we need to match a detected line with a ground-truth arc, we use the same procedure with the labels d and g interchanged.

6.4 Arc-Circle Matching

The following steps describe how we compute $MatchScore(i, j)$ for an arc d in the *D-entity-list* and a circle g in the *G-entity-list*. Let C_d and C_g be the centers of d and g, and let R_d and R_g be the two radii.

Step 1: We compute the projection of arc d onto the circle g, $proj(d, g)$, which is that part of the circle g that falls within the angular extent formed by the arc d and its center C_d.

Step 2: If $proj(d, g)$ doesn't exist, we set $MatchScore(i, j)$ to zero and skip the rest of the steps. Else, we create a straight line segment between the two end points of the arc d and call it L_d. Similarly, we create a straight line segment between the two end points of $proj(d, g)$ and call it L_{dg}.

Step 3: We apply the line-line matching criteria to the pair L_d and L_{dg}. $MatchScore(i, j)$ is set to the result of the line to line match score times F, an adjustment factor. Here, F is the inside angle of the arc d (in degrees) divided by 360.

For matching a detected circle with a ground-truth arc, we use the same procedure with the labels d and g interchanged.

6.5 Circle-Circle Matching Algorithm and Criteria

To compute the $MatchScore$ for a detected circle d and a ground-truth circle g as follows: Here, C_d and C_g are the centers, and R_d and R_g are the radii, of d and g, respectively (see Fig. 12).

Step 1: If the two circles, d and g, are identical (with the same centers and the same radii), we set $MatchScore(i, j)$ to 1 and skip the following steps.

Step 2: We compute the point-to-point distance, $ppDist(C_d, C_g)$, between the two centers C_d and C_g. If this distance is greater than θ_{CC}, we set $MatchScore(i, j)$ to zero and skip the following steps.

Step 3: We compute the absolute difference between the two radii, $RadiusDiff(R_d, R_g) = |R_d - R_g|$. If this distance is greater than the threshold θ_{RR}, we set $MatchScore(i, j)$ to zero and skip the following steps.

Step 4: We compute the ratio of these two radii, $RadiusRatio(R_d, R_g) = \min(R_d, R_g)/\max(R_d, R_g)$. If this ratio is smaller than a preset threshold (0.85 or as specified by the user), we set $MatchScore(i, j)$ to zero. Otherwise, we set $MatchScore(i, j)$ to $RadiusRatio(R_d, R_g)$.

TABLE 2
Test Reults on Image ds29 (Total Entities = 368)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.750	0.250	0.526	0.474	586	0.5477
	MDUS	0.758	0.242	0.389	0.611	284	0.3472
	VPstudio	0.772	0.228	0.495	0.505	380	0.4130
0.55	I/Vector	0.715	0.285	0.566	0.434	594	0.5551
	MDUS	0.758	0.242	0.387	0.613	286	0.3496
	VPstudio	0.764	0.236	0.498	0.502	388	0.4217
0.6	I/Vector	0.712	0.288	0.567	0.433	596	0.5570
	MDUS	0.758	0.242	0.378	0.622	294	0.3594
	VPstudio	0.761	0.239	0.500	0.500	390	0.4239
0.65	I/Vector	0.712	0.288	0.567	0.433	596	0.5570
	MDUS	0.755	0.245	0.378	0.622	298	0.3643
	VPstudio	0.761	0.239	0.500	0.500	390	0.4239
0.7	I/Vector	0.704	0.296	0.574	0.426	606	0.5664
	MDUS	0.753	0.247	0.382	0.618	298	0.3643
	VPstudio	0.755	0.245	0.505	0.495	398	0.4326
0.75	I/Vector	0.677	0.323	0.598	0.402	620	0.5794
	MDUS	0.750	0.250	0.382	0.618	304	0.3716
	VPstudio	0.750	0.250	0.511	0.489	412	0.4478
0.8	I/Vector	0.666	0.334	0.603	0.397	642	0.6000
	MDUS	0.745	0.255	0.387	0.613	324	0.3961
	VPstudio	0.758	0.242	0.513	0.487	422	0.4587
0.85	I/Vector	0.628	0.372	0.628	0.372	668	0.6243
	MDUS	0.728	0.272	0.396	0.604	340	0.4156
	VPstudio	0.739	0.261	0.525	0.475	436	0.4739
0.9	I/Vector	0.628	0.372	0.638	0.362	692	0.6467
	MDUS	0.717	0.283	0.427	0.573	366	0.4474
	VPstudio	0.726	0.274	0.543	0.457	466	0.5065

6.6 Text-Text Matching Algorithm and Criteria

A text area is represented by the rectangular bounding box of the text and its orientation. The rectangular box is described by the x and y coordinates of any two opposite corners of the rectangle and the orientation of the rectangle.

Let d be a text entity in the *D-entity-list* and let g be a text entity in the *G-entity-list*. Let $d1$ and $d2$ be the two opposite corners of d and let $g1$ and $g2$ be the two opposite corners of g (see Fig. 13).

For computing the $MatchScore$ for d and g, we follow the steps below:

Step 1: If d and g are identical (a perfect match), we set $MatchScore(i, j)$ to 1 and skip the following steps.

Step 2: Let d_{mid} be the midpoint of the diagonal line of d and let g_{mid} be the midpoint of the diagonal line of g. Without loss of generality, let $ppDist(d1, d2) > ppDist(g1, g2)$. That is, d has longer diagonal than that of g. We construct a circle centered at d_{mid} with radius equal to the diagonal of d. If both the corner points of g, $g1$, and $g2$, are outside of this circle (this means that there is no overlap between d and g), we set $MatchScore(i, j)$ to zero, and skip the following steps. This step is designed to limit the search space.

Step 3: Next, we compute the other two opposite corners of d and g so that each of the text boxes is now represented as a rectangle.

Step 4: We compute $d \cap g$, the intersection of d and g. If $d \cap g$ is empty, we set $MatchScore(i, j)$ to zero. Otherwise, we compute the area of d, g, and $d \cap g$. And, we set $MatchScore(i, j)$ to

$$\frac{area(d \cap g)}{\max(area(d), area(g))}.$$

We do not compare the orientation of the two text boxes, d and g. As stated earlier, we use the respective orientations only to construct the text boxes. In particular, it is perfectly fine for the orientation of a text box to be off by a multiple of 90°. We are only interested in comparing the bounding boxes and not in the true direction of the text.

For the software implementation of text-to-text matching, the algorithm for computing the area of intersection of two rectangles was taken from [14].

7 Performance Metrics

Performance measurements for a recognition system can be formulated using a linear combination of some or all of the matching results—the counts of matches, false-alarms, and misses. Let $one2one$ be the count of the one-to-one matches,

TABLE 3
Test Results on Image ds30 (Total Entities = 443)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.754	0.246	0.538	0.462	648	0.5294
	MDUS	0.673	0.327	0.420	0.580	399	0.4222
	VPstudio	0.745	0.255	0.594	0.406	613	0.4980
0.55	I/Vector	0.729	0.271	0.565	0.435	654	0.5343
	MDUS	0.679	0.321	0.424	0.576	403	0.4265
	VPstudio	0.749	0.251	0.594	0.406	625	0.5077
0.6	I/Vector	0.725	0.275	0.565	0.435	660	0.5392
	MDUS	0.677	0.323	0.426	0.574	405	0.4286
	VPstudio	0.745	0.255	0.599	0.401	625	0.5077
0.65	I/Vector	0.722	0.278	0.567	0.433	668	0.5458
	MDUS	0.677	0.323	0.424	0.576	407	0.4307
	VPstudio	0.763	0.237	0.598	0.402	631	0.5126
0.7	I/Vector	0.729	0.271	0.571	0.429	682	0.5572
	MDUS	0.695	0.305	0.424	0.576	411	0.4349
	VPstudio	0.770	0.230	0.598	0.402	635	0.5158
0.75	I/Vector	0.720	0.280	0.583	0.417	704	0.5752
	MDUS	0.695	0.305	0.426	0.574	415	0.4392
	VPstudio	0.763	0.237	0.600	0.400	649	0.5272
0.8	I/Vector	0.707	0.293	0.597	0.403	716	0.5850
	MDUS	0.688	0.312	0.438	0.562	427	0.4519
	VPstudio	0.763	0.237	0.602	0.398	659	0.5353
0.85	I/Vector	0.691	0.309	0.607	0.393	732	0.5980
	MDUS	0.673	0.327	0.460	0.540	441	0.4667
	VPstudio	0.738	0.262	0.617	0.383	679	0.5516
0.9	I/Vector	0.677	0.323	0.615	0.385	750	0.6127
	MDUS	0.650	0.350	0.484	0.516	459	0.4857
	VPstudio	0.655	0.345	0.637	0.363	697	0.5662

$g\_one2many$ be the count of the g_one-to-many matches, $g\_many2one$ be the count of the g_many-to-one matches, $d\_one2many$ be the count of the d_one-to-many matches, $d\_many2one$ be the count of the d_many-to-one matches, $false\_alarms$ be the count of the false-alarms, $misses$ be the count of the misses, N be the count of entities in the ground-truth file, and M be the count of entities in the result file (the detected entities).

Below, we define several system performance metrics. For the purpose of generating the performance measurements that are tabulated later in the paper, all the weights, ws, used below were set to an ad hoc value of one (as mentioned below). In order to establish the most useful performance metrics, the relative significance of each weight needs to be researched further. We want the performance metrics to not only tell us how well a recognition system does but also to give us an idea of the cost of correcting the recognition results. Therefore, the best way to determine the weights is perhaps empirical.

Detection rate

$$DetectionRate = w_1 \cdot \frac{one2one}{N} + w_2 \cdot \frac{g\_one2many}{N} + w_3 \cdot \frac{g\_many2one}{N}.$$

$DetectionRate$ is, roughly, the percentage of ground-truth entities that are detected by the recognition system. In our implementation, we set w_1, w_2, and w_3 to 1.

Missed Detection Rate

$$MissedDetectionRate = \frac{misses}{N}.$$

$MissedDetectionRate$ is the percentage of ground-truth entities which are not detected by the recognition system. Note that $DetectionRate$ and $MissedDetectionRate$ may not necessary add up to one, because of the factors involved in the computation of $DetectionRate$.

False-alarm Rate

$$FalseAlarmRate = \frac{false\_alarms}{M}.$$

$FalseAlarmRate$ is the percentage of detected entities produced by the system that do not match with any entity in the ground-truth.

Recognition Accuracy

$$RecognitionAccuracy = w_4 \cdot \frac{one2one}{M} + w_5 \cdot \frac{d\_one2many}{M} + w_6 \cdot \frac{d\_many2one}{M}.$$

TABLE 4
Test Results on Image ds31 (Total Entities = 626)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.716	0.284	0.483	0.517	644	0.4259
	MDUS	0.719	0.281	0.323	0.677	412	0.3234
	VPstudio	0.757	0.243	0.513	0.487	659	0.4189
0.55	I/Vector	0.722	0.278	0.480	0.520	650	0.4299
	MDUS	0.719	0.281	0.323	0.677	412	0.3234
	VPstudio	0.757	0.243	0.513	0.487	659	0.4189
0.6	I/Vector	0.719	0.281	0.484	0.516	654	0.4325
	MDUS	0.716	0.284	0.324	0.676	418	0.3281
	VPstudio	0.757	0.243	0.510	0.490	663	0.4215
0.65	I/Vector	0.717	0.283	0.479	0.521	670	0.4431
	MDUS	0.716	0.284	0.321	0.679	422	0.3312
	VPstudio	0.757	0.243	0.508	0.492	667	0.4240
0.7	I/Vector	0.716	0.284	0.475	0.525	678	0.4484
	MDUS	0.716	0.284	0.321	0.679	422	0.3312
	VPstudio	0.754	0.246	0.509	0.491	673	0.4278
0.75	I/Vector	0.709	0.291	0.471	0.529	690	0.4563
	MDUS	0.714	0.286	0.323	0.677	424	0.3328
	VPstudio	0.749	0.251	0.512	0.488	679	0.4317
0.8	I/Vector	0.690	0.310	0.495	0.505	714	0.4722
	MDUS	0.706	0.294	0.330	0.670	446	0.3501
	VPstudio	0.748	0.252	0.515	0.485	689	0.4380
0.85	I/Vector	0.677	0.323	0.497	0.503	734	0.4854
	MDUS	0.688	0.312	0.347	0.653	468	0.3673
	VPstudio	0.732	0.268	0.521	0.479	697	0.4431
0.9	I/Vector	0.658	0.342	0.512	0.488	756	0.5000
	MDUS	0.658	0.342	0.383	0.617	512	0.4019
	VPstudio	0.730	0.270	0.523	0.477	703	0.4469

$RecognitionAccuracy$ indicates, roughly, the percentage of detected entities within the result file that have their match in the ground-truth entities. Thus, one can consider $Recognition\ Accuracy$ as a measurement of the overall accuracy of a recognition system. In our implementation, we set w_4, w_5, and w_6 to 1.

Post-editing Cost

$$\begin{aligned} EditCost = {} & w_7 \cdot false\_alarms + w_8 \cdot misses \\ & + w_9 \cdot g\_one2many + w_{10} \cdot g\_many2one \\ & + w_{11} \cdot d\_one2many + w_{12} \cdot d\_many2one. \end{aligned}$$

$EditCost$ is the cost estimate for human post-editing effort to clean-up the recognition result. It should be clear that a higher $EditCost$ requires a higher post-editing effort. Entities missing from the result file need to be added and the false-alarms need to be removed. Moreover, for each one-to-many and many-to-one match, one needs to remove the many partially matching entities and add the real entity. w_7 is the factor for one deletion effort and w_8 is the factor for one insertion effort. The weights w_9 through w_{12} are more complex; they depend on the cost of merging several detected entities into one and splitting a detected entity into several entities, respectively. These factors should depend on the post-editing tool one uses. In our implementation, we set w_7 through w_{12} to one.

EditCost Index

$$\begin{aligned} & EditCostIndex = \\ & (false\_alarms + misses + g\_one2many \\ & + g\_many2one + d\_one2many + d\_many2one)/(N + M). \end{aligned}$$

$EditCostIndex$ is one measure that combines all the above metrics into an estimate of the normalized post-editing cost. The index can have values between zero and one. The lower the value of the index, the less post-editing one has to do. It should be noted that

$$\begin{aligned} & (false\_alarms + misses + g\_one2many \\ & + g\_many2one + d\_one2many + d\_many2one)/(N + M) + \\ & (2 \cdot one2one)/(N + M) = 1. \end{aligned}$$

In other words,

$$EditCostIndex = 1 - \frac{2 \cdot one2one}{N + M}.$$

As before, it is possible to assign weights to the different components of the $EditCostIndex$ to account for the amount of effort required to correct the respective types of mistakes.

TABLE 5
Test Results on Image ds32 (Total Entities = 513)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.756	0.244	0.369	0.631	427	0.3659
	MDUS	0.768	0.232	0.261	0.739	275	0.2632
	VPstudio	0.821	0.179	0.400	0.600	385	0.3179
0.55	I/Vector	0.750	0.250	0.373	0.627	435	0.3728
	MDUS	0.766	0.234	0.261	0.739	279	0.2670
	VPstudio	0.821	0.179	0.398	0.602	387	0.3196
0.6	I/Vector	0.747	0.253	0.375	0.625	441	0.3779
	MDUS	0.762	0.238	0.265	0.735	283	0.2708
	VPstudio	0.817	0.183	0.401	0.599	391	0.3229
0.65	I/Vector	0.743	0.257	0.379	0.621	457	0.3916
	MDUS	0.762	0.238	0.263	0.737	299	0.2861
	VPstudio	0.823	0.177	0.401	0.599	395	0.3262
0.7	I/Vector	0.741	0.259	0.376	0.624	469	0.4019
	MDUS	0.756	0.244	0.269	0.731	311	0.2976
	VPstudio	0.823	0.177	0.401	0.599	395	0.3262
0.75	I/Vector	0.713	0.287	0.414	0.586	487	0.4173
	MDUS	0.752	0.248	0.274	0.726	319	0.3053
	VPstudio	0.823	0.177	0.403	0.597	399	0.3295
0.8	I/Vector	0.708	0.292	0.425	0.575	501	0.4293
	MDUS	0.754	0.246	0.280	0.720	331	0.3167
	VPstudio	0.821	0.179	0.407	0.593	409	0.3377
0.85	I/Vector	0.694	0.306	0.433	0.567	517	0.4430
	MDUS	0.735	0.265	0.301	0.699	353	0.3378
	VPstudio	0.815	0.185	0.413	0.587	413	0.3410
0.9	I/Vector	0.665	0.335	0.457	0.543	555	0.4756
	MDUS	0.715	0.285	0.323	0.677	379	0.3627
	VPstudio	0.797	0.203	0.428	0.572	427	0.3526

8 Benchmark Specifications

The benchmark for evaluating the performance of graphics recognition systems consists of the following components:

- File format specification for ground-truth vector data,
- A set of training images with ground-truth vector data,
- A set of test images with ground-truth vector data, and
- Software for performance evaluation.

The software operates on the recognition results of any graphics recognition system, compares them with the ground-truth data using the matching algorithms of Section 6, computes match counts using the method of Section 5, and computes the performance metrics proposed in Section 7. In order to compare several recognition systems, one can use this benchmark on all the systems and compare the performance measures by tabulating or plotting them.

The evaluation software was written in the C programming language. It has been compiled and tested on Sun SPARCstations running Solaris 2.5.1, PCs running Linux kernel 2.0.32 or Windows 95, and Silicon Graphics Indy running Irix 6.2.

Currently, the data set contains only synthetically generated images. These images were obtained by rendering real CAD files into raster format. The CAD-to-raster conversion software was specially developed for this benchmark. In addition to doing the conversion, this software allows one to add randomness to the attributes of the vector elements being rendered. The CAD-to-raster conversion software was written in Perl and C programming languages. It uses several public domain software components [15], [16], [17]. This software was compiled and tested on Sun SPARCstations running Solaris 2.5.1 and PCs running Linux kernel 2.0.32. More details about the conversion tools, the rendering process, and preparation of the training and test data can be found in [18].

The complete benchmark can be downloaded from [6].

8.1 Data Set

In order to do a meaningful comparison of graphics recognition systems, one must use real images of scanned line drawing documents. However, that requires a very large effort to carefully create the ground truth data for each image. Moreover, using real scanned images, it is not possible to control the vector entities in various ways so as to test the breaking points of the recognition systems. Therefore, in the current benchmark, we decided to use only synthetic images. The obvious advantages of

TABLE 6
Test Results on Image ds09 (Total Entities = 447)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.790	0.210	0.392	0.608	393	0.3887
	MDUS	0.671	0.329	0.422	0.578	389	0.4215
	VPstudio	0.776	0.224	0.461	0.539	422	0.4027
0.55	I/Vector	0.787	0.213	0.392	0.608	393	0.3887
	MDUS	0.669	0.331	0.426	0.574	389	0.4215
	VPstudio	0.774	0.226	0.459	0.541	428	0.4084
0.6	I/Vector	0.787	0.213	0.388	0.612	397	0.3927
	MDUS	0.669	0.331	0.426	0.574	389	0.4215
	VPstudio	0.774	0.226	0.458	0.542	430	0.4103
0.65	I/Vector	0.787	0.213	0.383	0.617	407	0.4026
	MDUS	0.669	0.331	0.426	0.574	389	0.4215
	VPstudio	0.774	0.226	0.456	0.544	432	0.4122
0.7	I/Vector	0.785	0.215	0.376	0.624	415	0.4105
	MDUS	0.671	0.329	0.422	0.578	395	0.4280
	VPstudio	0.772	0.228	0.456	0.544	432	0.4122
0.75	I/Vector	0.779	0.221	0.390	0.610	421	0.4164
	MDUS	0.667	0.333	0.424	0.576	401	0.4345
	VPstudio	0.767	0.233	0.459	0.541	444	0.4237
0.8	I/Vector	0.761	0.239	0.406	0.594	433	0.4283
	MDUS	0.660	0.340	0.431	0.569	407	0.4410
	VPstudio	0.761	0.239	0.463	0.537	452	0.4313
0.85	I/Vector	0.745	0.255	0.422	0.578	455	0.4500
	MDUS	0.649	0.351	0.445	0.555	419	0.4540
	VPstudio	0.752	0.248	0.471	0.529	456	0.4351
0.9	I/Vector	0.709	0.291	0.459	0.541	477	0.4718
	MDUS	0.626	0.374	0.471	0.529	435	0.4713
	VPstudio	0.745	0.255	0.478	0.522	468	0.4466

synthetically generated images are 1) it is easy to create the ground truth files and 2) one can have complete control over the individual graphical entities and the image rendering process. Although it is always desirable to eventually test with real images, synthetically generated images offer a good starting point in the evaluation. Synthetic data is invaluable for studying the breaking points of recognition systems. Further, synthetic images and ground truth are very inexpensive to generate.

We chose the very practical approach of using *real* CAD drawings to create synthetic images and ground truth data. We obtained the CAD drawings from the University of Washington's UW-III document image database [19]. The selected CAD drawings are complex, "real life" archived drawings.

As one may notice in Section 6, the current matching algorithms do not handle graphical objects like arrowheads, hatched areas, etc. Therefore, during the CAD-to-image rendering process, we filtered out all entities except lines, arcs, circles, and text. Some artificial distortions were added randomly to the vector entities to help make the resulting images resemble real images. The distortions were simple, such as differing thickness of different lines, arcs, and circles; varying length of dashes and gaps in dashed lines; and the varying orientation, size, and stroke width of text (this causes some text to touch or overlap with graphics—a very real problem in real images). We used clean images of realistic CAD drawings (which are quite complex), degraded by manipulating the vector entities, but not degraded at the image level. Each of the resulting images contains over 500 graphical entities (see Table 1).

The "vector" distortions were not validated empirically. However, they are the kind of vector distortions that are observed very often in manually drafted drawings. For example, when the text is "drafted" manually on a paper drawing, there is no guarantee that the baseline of the text is perfectly horizontally or vertically oriented. This is emulated by adding Gaussian noise to the orientation of text entities. In CAD drawings, almost all lines have zero thickness. In fact, in AutoCAD [20], the line entity does not have a thickness attribute. In manually drafted drawings, however, lines are drawn with pens of varying thickness. We emulate this by adding thickness to the lines and by making this thickness a normal variate.

In future, we will impose image noise in addition to the vector noise. Significant work has been done on building and validating image noise models for typical document (page) images [21], [22], [23]. However, these models account for only some of the noise found in scanned images of aging drawings that were drafted on paper or parchment. More severe image distortions arise from scanning blue-

TABLE 7
Test Results on Image ds07 (Total Entities = 1,360)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.677	0.323	0.388	0.612	1119	0.4094
	MDUS	0.555	0.445	0.496	0.504	1391	0.5184
	VPstudio	0.701	0.299	0.466	0.534	1291	0.4331
0.55	I/Vector	0.673	0.327	0.391	0.609	1139	0.4168
	MDUS	0.556	0.444	0.496	0.504	1393	0.5192
	VPstudio	0.700	0.300	0.468	0.532	1293	0.4337
0.6	I/Vector	0.670	0.330	0.398	0.602	1163	0.4255
	MDUS	0.559	0.441	0.497	0.503	1405	0.5237
	VPstudio	0.701	0.299	0.471	0.529	1315	0.4411
0.65	I/Vector	0.665	0.335	0.399	0.601	1181	0.4321
	MDUS	0.559	0.441	0.497	0.503	1405	0.5237
	VPstudio	0.697	0.303	0.474	0.526	1325	0.4445
0.7	I/Vector	0.660	0.340	0.403	0.597	1203	0.4402
	MDUS	0.559	0.441	0.497	0.503	1405	0.5237
	VPstudio	0.691	0.309	0.478	0.522	1343	0.4505
0.75	I/Vector	0.653	0.347	0.409	0.591	1223	0.4475
	MDUS	0.558	0.442	0.497	0.503	1407	0.5244
	VPstudio	0.689	0.311	0.480	0.520	1349	0.4525
0.8	I/Vector	0.646	0.354	0.414	0.586	1247	0.4563
	MDUS	0.557	0.443	0.497	0.503	1411	0.5259
	VPstudio	0.685	0.315	0.481	0.519	1365	0.4579
0.85	I/Vector	0.629	0.371	0.434	0.566	1297	0.4746
	MDUS	0.554	0.446	0.500	0.500	1419	0.5289
	VPstudio	0.679	0.321	0.486	0.514	1381	0.4633
0.9	I/Vector	0.618	0.382	0.443	0.557	1351	0.4943
	MDUS	0.542	0.458	0.525	0.475	1515	0.5647
	VPstudio	0.674	0.326	0.495	0.505	1399	0.4693

prints. Much work needs to be done in the area of image noise models for line drawings.

We selected four types of CAD drawings from the UW-III database—mechanical, architectural, and two distinct types of utility drawings. The images are carefully partitioned into two sets so that the images in the training set and the testing set have similar characteristics. Four training images are shown in Figs. 14, 15, 16, and 17. The test images have not been made public since we intend to reuse them in future graphics recognition benchmarks.

8.2 Image File Format

Only bilevel images are used in this benchmark. The images are in TIFF 6.0 CCITT Group 4 format.

8.3 Vector File Format

The most popular public format for vector files is the AutoCAD DXF format [20]. This format is too complex for the quantitative evaluation of recognition results, especially given the types of graphical entities considered in the benchmark. In order to make the evaluation simple, we specify a much simpler vector file format (the VEC format).

VEC files are ASCII files. The VEC format has only four graphical primitives—lines, arcs, circles, and text regions. Below are the attributes associated with each of the four primitives in the above order. Each primitive is contained in a single row in the VEC file. All dimensions are interpreted as floating point values. The first row specifies the `xsize` and `ysize` of the image. The `x` axis points rightward and the `y` axis points downward. The units of `x` and `y` are pixels.

```
%VEC-1.0 xsize ysize [dpi]
L C|D x1 y1 x2 y2 width
A C|D xcenter ycenter radius start_angle end_angle width
C C|D xcenter ycenter radius width
T x1 y1 x2 y2 orientation fontHeight fontWidthFactor \
    fontStrokeWidth %TEXT
```

The first line is the VEC file indicator, followed by a list of entity descriptions. The first letter of each entity description stands for the entity type: L for line, A for arc, C for circle, and T for text. The second letter indicated whether the entity is continuous (C) or dashed (D). The remaining fields are the attributes of the respective entities. See [24], [18] for detailed specification of the VEC file format.

9 The Graphics Recognition Contest: Evaluation of Results and Performance Analysis

This benchmark was used in the Second International Graphics Recognition Contest held at Nancy, France, in August 1997 [24], [25]. Three systems participated in the

TABLE 8
Test Results on Image ds35 (Total Entities = 639)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.875	0.125	0.166	0.834	236	0.1774
	MDUS	0.402	0.598	0.552	0.448	699	0.5763
	VPstudio	0.887	0.113	0.301	0.699	397	0.2673
0.55	I/Vector	0.875	0.125	0.168	0.832	240	0.1805
	MDUS	0.405	0.595	0.552	0.448	703	0.5796
	VPstudio	0.873	0.127	0.312	0.688	415	0.2795
0.6	I/Vector	0.875	0.125	0.164	0.836	246	0.1850
	MDUS	0.405	0.595	0.552	0.448	703	0.5796
	VPstudio	0.861	0.139	0.311	0.689	449	0.3024
0.65	I/Vector	0.870	0.130	0.168	0.832	252	0.1895
	MDUS	0.405	0.595	0.552	0.448	703	0.5796
	VPstudio	0.834	0.166	0.220	0.780	479	0.3226
0.7	I/Vector	0.862	0.138	0.175	0.825	262	0.1970
	MDUS	0.405	0.595	0.552	0.448	703	0.5796
	VPstudio	0.825	0.175	0.343	0.657	489	0.3293
0.75	I/Vector	0.840	0.160	0.210	0.790	282	0.2120
	MDUS	0.405	0.595	0.540	0.460	717	0.5911
	VPstudio	0.818	0.182	0.353	0.647	491	0.3306
0.8	I/Vector	0.820	0.180	0.223	0.777	332	0.2496
	MDUS	0.393	0.607	0.568	0.432	729	0.6010
	VPstudio	0.797	0.203	0.379	0.621	523	0.3522
0.85	I/Vector	0.790	0.210	0.247	0.753	372	0.2797
	MDUS	0.365	0.635	0.601	0.399	767	0.6323
	VPstudio	0.762	0.238	0.409	0.591	561	0.3778
0.9	I/Vector	0.753	0.247	0.282	0.718	420	0.3158
	MDUS	0.341	0.659	0.627	0.373	801	0.6603
	VPstudio	0.740	0.260	0.428	0.572	585	0.3939

contest; two were commercial products (I/Vector [7] and VPstudio [8]) and one was a research prototype from a university (MDUS [26], [9], [27]). The test images used in this benchmark consist of four mechanical drawings, one architectural drawing, two utility drawings, and one structural drawing, a total of eight test images. Each of the three participants was given an opportunity to optimize the parameters of the respective system for each type of drawing, based on the training images. Then, all the systems were tested on all the test images. For each test image, recognition results of the systems were compared with the ground-truth in order to obtain the measures of performance. The performance metrics of each system were then compared with others.

Recall that the performance evaluator uses an acceptance threshold to determine whether a pair of entities matches (a pair is said to match when its *MatchScore* is equal to or higher than the acceptance threshold.) The matching criteria for a pair of entities defined in Section 5 is roughly a similarity measurement. When the acceptance threshold is set high, the evaluator accepts only those pairs that are very similar (having high matching scores). When we lower the acceptance threshold, the evaluator lowers its matching requirement. We expect that, with a high acceptance threshold, good systems can score better in their performance measurements. We are additionally interested in discovering the trends of system performance with respect to changes in the acceptance threshold. We propose that, for good recognition systems, lowering the acceptance threshold may improve the performance a little, but not drastically. On the other hand, for the not-so-good systems, the performance measurements may improve greatly when the evaluator's acceptance threshold is set lower. Thus, varying the acceptance threshold in the evaluation may reveal the stability of a recognition system.

With the above concepts in mind, nine acceptance thresholds were used in the performance evaluation—from 0.5 to 0.9, in steps of 0.05. That is, for each recognition result file produced by a system, we obtained nine sets of matching counts using these nine acceptance thresholds. In Tables 2, 3, 4, 5, 6, 7, 8, and 9, we list the performance measurements of Section 7 for the three systems with respect to the nine thresholds.

9.1 Analysis of Performance Characteristics

We designed the benchmark to compare the performance of different graphics recognition systems. We hoped to study the performance characteristics of the systems by changing the acceptance threshold of our evaluator. Below, we describe the observed characteristics for the three systems that participated in the second international graphics recognition contest [24].

TABLE 9
Test Results on Image ds36 (Total Entities = 1,028)

Acceptance Threshold	System Name	Detection Rate	Missed Detection Rate	False-alarm Rate	Recognition Accuracy	Edit Cost	EditCost Index
0.5	I/Vector	0.706	0.294	0.214	0.786	585	0.2983
	MDUS	0.672	0.328	0.243	0.757	603	0.3120
	VPstudio	0.713	0.287	0.320	0.680	692	0.3327
0.55	I/Vector	0.715	0.285	0.211	0.789	601	0.3065
	MDUS	0.684	0.316	0.240	0.760	627	0.3244
	VPstudio	0.724	0.276	0.322	0.678	712	0.3423
0.6	I/Vector	0.715	0.285	0.212	0.788	609	0.3106
	MDUS	0.686	0.314	0.239	0.761	633	0.3275
	VPstudio	0.722	0.278	0.323	0.677	718	0.3452
0.65	I/Vector	0.713	0.287	0.211	0.789	623	0.3177
	MDUS	0.678	0.322	0.241	0.759	649	0.3357
	VPstudio	0.722	0.278	0.324	0.676	722	0.3471
0.7	I/Vector	0.706	0.294	0.218	0.782	637	0.3248
	MDUS	0.671	0.329	0.251	0.749	683	0.3533
	VPstudio	0.720	0.280	0.329	0.671	732	0.3519
0.75	I/Vector	0.703	0.297	0.223	0.777	659	0.3361
	MDUS	0.666	0.334	0.263	0.737	709	0.3668
	VPstudio	0.713	0.287	0.331	0.669	754	0.3625
0.8	I/Vector	0.686	0.314	0.233	0.767	695	0.3544
	MDUS	0.655	0.345	0.271	0.729	767	0.3968
	VPstudio	0.706	0.294	0.344	0.656	798	0.3837
0.85	I/Vector	0.685	0.315	0.232	0.768	755	0.3850
	MDUS	0.640	0.360	0.283	0.717	827	0.4278
	VPstudio	0.704	0.296	0.349	0.651	830	0.3990
0.9	I/Vector	0.670	0.330	0.241	0.759	809	0.4125
	MDUS	0.625	0.375	0.309	0.691	927	0.4796
	VPstudio	0.695	0.305	0.365	0.635	888	0.4269

One of the most common techniques used to compare pattern classifiers is the Receiver Operating Characteristic (ROC) curve [28]. The curve is obtained by plotting the rate of true positives vs. the rate of false alarms. The area under the curve is taken as a measure of the overall goodness of the classifier. It is natural to attempt to apply the ROC technique to our current problem. However, there are some significant differences between traditional classifiers and graphics recognition systems. Given an input, a classifier determines whether or not the input belongs to the appropriate class. Only one answer is generated for a given input pattern. This answer is accompanied by a confidence level. By varying the confidence level at which one accepts or rejects the classifier output, one can plot a true positive vs. false positive curve. In contrast, graphics recognition methods can generate many output entities corresponding to the image of one graphical entity. For example, two crossing lines may be recognized as two lines, three lines, or four lines. Further, due to image noise, graphics recognition systems often fragment lines even where there are no crossing points or junctions among lines. This creates a problem in applying the ROC technique as there is no one-to-one correspondence between the image of a graphical entity and the recognized result. Moreover, graphics recognition systems do not use statistical methods for recognizing entities; they use variants of line following. Therefore, these systems do not generate confidence levels for the results. Consequently, one cannot generate an ROC curve by varying the acceptable confidence level. Due to the above reasons, we adopt a different technique to compare graphics recognition systems.

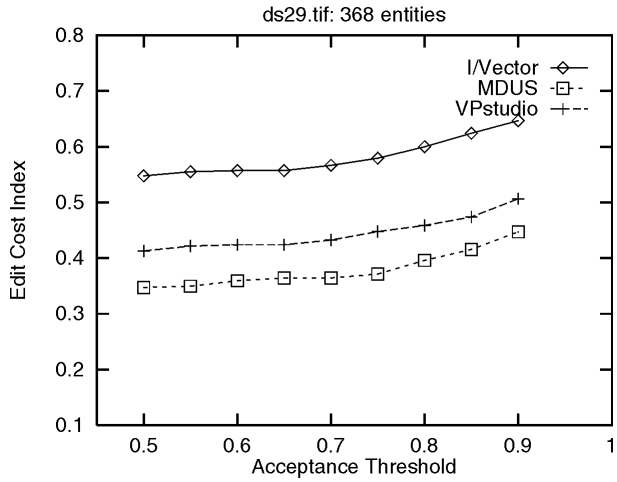

Fig. 18. Performance curves of the systems for the image ds29.tif (a mechnical drawing).

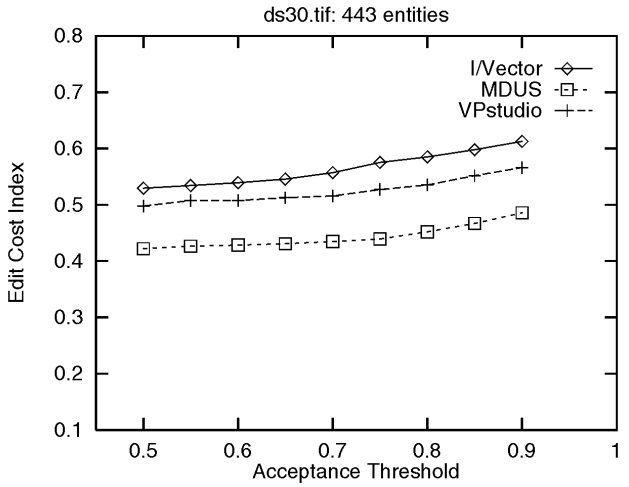

Fig. 19. Performance curves of the systems for the image dds30.tif (a mechanical drawing).

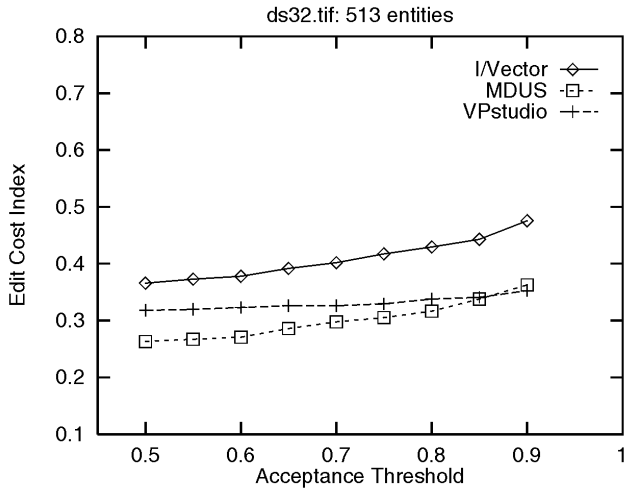

Fig. 21. Performance curves of the systems for the image ds32.tif (a mechanical drawing).

In an earlier report on the graphics recognition contest [24], we plotted the counts of the false-alarms vs. the misses for the various settings of the acceptance threshold. These earlier plots gave us insights into the internal behavior of the systems. For each system, these plots told us how fast the counts of misses and false alarms rose as we raised the acceptance threshold. They did not give us an idea of the overall post-editing cost for these systems. At the time, we had not formulated the *EditCost Index*. The *EditCost Index* proposed here gives us a very powerful way of comparing the overall post-editing cost for the recognition results produced by various systems. It captures all the other performance metrics of Section 7 into one measure. All of the metrics are tabulated in Tables 2, 3, 4, 5, 6, 7, 8, and 9. In Figs. 18, 19, 20, 21, 22, 23, 24, and 25, we plot the *EditCost Index* versus the acceptance threshold.

From these tables and plots, we observe the following:

- In general, all three curves in each of the plots show a gradual upward trend. That is, as the acceptance threshold is increased, all three systems produce a larger combination of misses, false-alarms, and partial matches. However, some systems exhibit less of an upward trend compared to others. For example, in Figs. 20, 21, 22, 23, the VPstudio system has significantly smaller increases in the *EditCost Index*, compared with the others two systems, as the acceptance threshold is increased.
- In the earlier report [24], we noted that points on the false-alarms vs. misses curves formed a tight cluster for the values of the acceptance threshold between 0.5 and 0.65. In the current plots, this translates into the observation that, for these lower values of the acceptance threshold, most curves are essentially flat.
- In most of the cases, all systems produce more false-alarms than misses. This may be partly due to one of the following reasons: 1) At present, the evaluator does not match any dashed entity to any solid entity. So, if a dashed-line in a test image is detected by a vectorization system as several little straight line segments, the evaluator produces counts of one miss (dashed-line) and several false-alarms (little line segments). 2) When a text string in a test image is not correctly detected as a text region, it is often "vectorized" into several small lines, arcs, etc. In this case, the evaluator currently produces counts of one miss (the missing text string) and several false-alarms (the little "vectors").
- In most cases, all the systems produce *EditCost* that is close to or greater than the number of ground-truth entities. At first look, this may be taken to mean that it is easier to create the drawing from scratch (using a CAD tool) rather than to convert it using a raster to vector conversion system followed by correction of the mistakes. In practice, one should not make this assumption without looking at the individual mistakes made by these systems and determining the effort required to correct the mistakes. For instance, in any CAD tool, it is quite

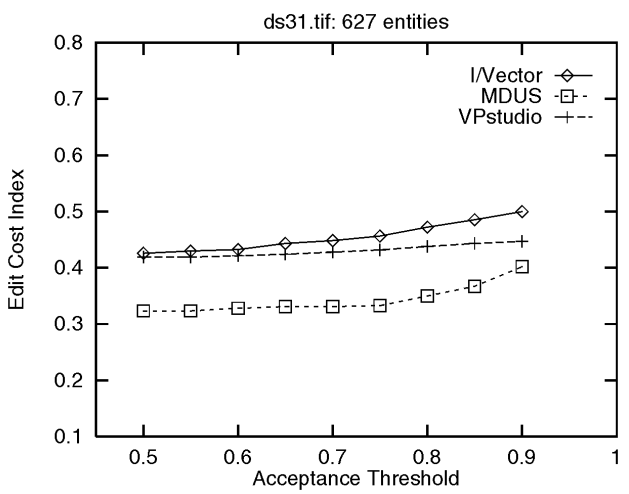

Fig. 20. Performance curves of the systems for the image ds31.tif (a mechanical drawing).

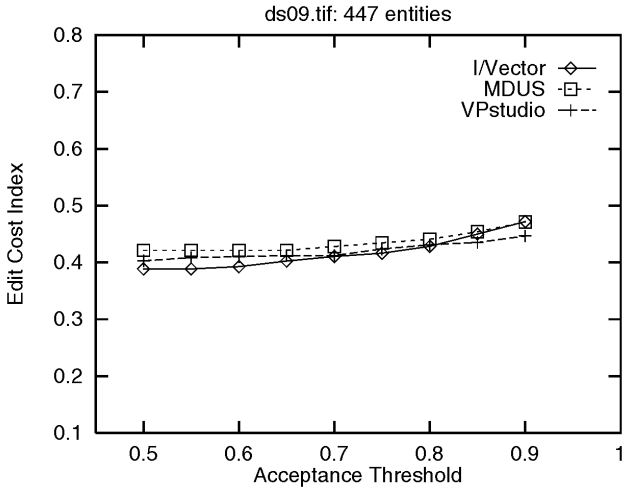

Fig. 22. Performance curves of the systems for the image ds09.tif (an architectural drawing).

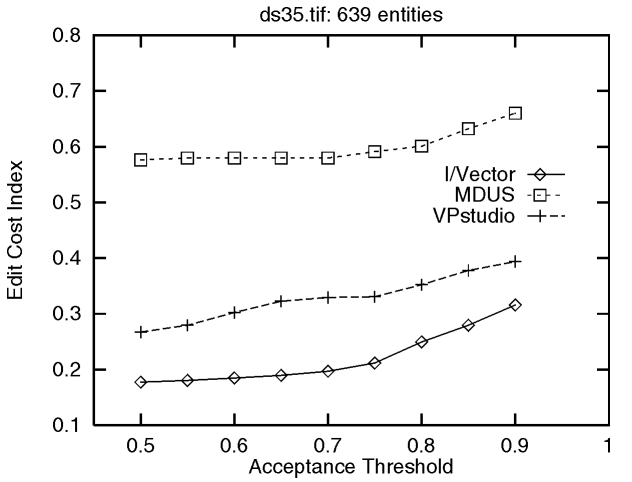

Fig. 24. Performance curves of the systems for the image ds35.tif (a utility drawing, type 2).

easy to select several adjacent false alarms and to delete them. To create individual entities corresponding to the misses takes more time. As noted above, all systems were found to generate more false alarms than misses. It should be clear that the empirical determination of the appropriate weights in Section 7 is quite important for proper evaluation of graphics recognition systems.

In addition to the above overall trends, we observed the following about the individual systems. The MDUS system, which was designed specifically for mechanical drawings, consistently produces the lowest *EditCost Index* for all the mechanical drawings in the benchmark. The VPstudio system consistently has the second lowest *EditCost Index* for all the mechanical drawings. Overall, the VPstudio system exhibits the smallest increase in the *EditCost Index* with increasing acceptance threshold. On two of the four nonmechanical drawings in the test set, the performance curves of the three systems are almost identical. On the remaining two test drawings, the MDUS system has significantly higher *EditCost Index* than the I/Vector and VPstudio systems.

From the above, we can clearly deduce that it helps to customize a recognition system using contextual rules and syntax. This is evidenced by the best performance of the MDUS system on all the mechanical drawings in the test set. Another obvious conclusion from the observations is the need for a much larger corpus of drawing images with ground truth. Due to insufficient data, it was hard to draw any conclusions about the systems on the nonmechanical drawings.

10 Current Limitations and Future Work

The benchmark used in the graphics recognition contest [24], and described above, limited itself to a quantitative evaluation of the automatic vectorization capability of graphics recognition systems. Several other constraints were imposed either due to lack of time and resources or

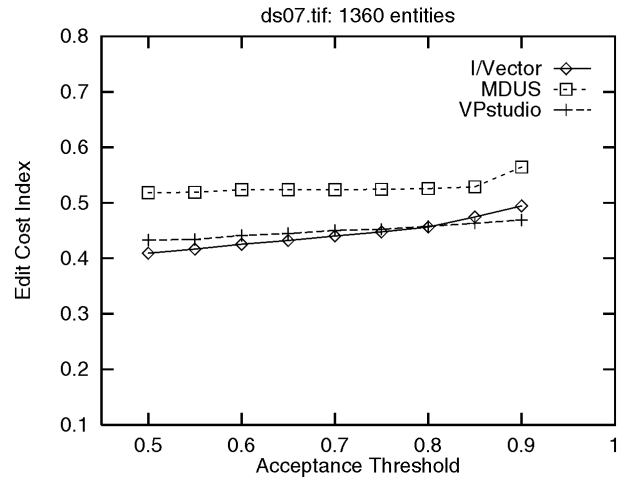

Fig. 23. Perofrmance curves of the systems for the image ds07.tif (a utility drawing, type 1).

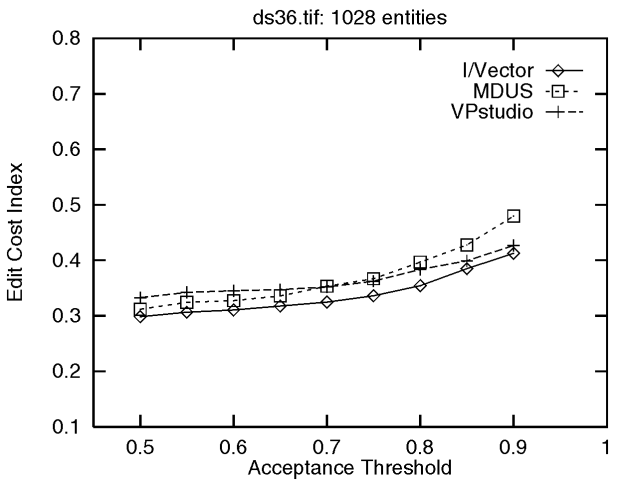

Fig. 25. Performance curves of the systems for the image ds36.tif (a structural drawing).

in order to keep the evaluation protocol simple. The primary constraints were as follows:

1. Only CAD images were used for both training and testing.
2. The only "noise" in the images was in the form of thickness of lines, length of dashes and gaps in dashed lines, and the orientation and size of text. No "image noise" was added.
3. We only tested at the image resolution of 200 dots per inch.
4. We only tested for detection of straight lines, arcs, circles, and text. Detection of polylines, dimensioning, objects, symbols, etc. was not tested.
5. Only one kind of dashed line was used. This was the simple dash-dash line.
6. No match was attempted between dashed entities and solid entities.

There are some known shortcomings in our evaluation process which we will address in the near future. If a vectorization system erroneously recognizes a dashed line as a sequence of short continuous lines, then our evaluation method assigns a single miss but a large number of false-alarms (because we do not attempt to match a dashed line with continuous line segments). We need to allow matching of dashed lines with several small line segments, but this should be penalized somewhat due to the fragmentation introduced.

If a text region is not correctly identified, then we assign a single miss accompanied with a large number of false-alarms. When recognition systems miss a text region, they invariably try to "vectorize" the region. The resulting short lines ("vectors") will count as false-alarms because we do not attempt to match a text area with any other type of entity. In order to correct the misinterpretation, one only needs to box a text region and mark it as text (we are not talking about OCR here; OCR is outside the scope of this benchmark). This is a very simple postprocessing operation. Therefore, this kind of error should not be penalized so heavily.

Gathering data to test and compare graphics recognition systems is very time consuming. This benchmark only used synthetic images with associated ground-truth. Future benchmarks should include synthetic images with image degradation (see Section 8.1) and real images with manually created ground-truth. The graphics recognition community needs to collaborate in building a database of images and ground-truth files.

The real strengths and weaknesses of a system are revealed by stress testing the system. We can accomplish this by testing the performance of a vectorization system with increasing image degradation and increasing image complexity. This should be attempted in a future benchmark.

Future benchmarks will hopefully attract participation from many more vectorization software companies. All the systems that we tested in this benchmark are among the best products or research prototypes available for vectorization. A larger number of systems in the benchmark will provide us broader trends and will give us a real assessment of the state of the technology.

REFERENCES

[1] B. Kong, I. Phillips, R. Haralick, A. Prasad, and R. Kasturi, "A Benchmark: Performance Evaluation of Dashed Line Detection Algorithms," *Graphics Recognition: Methods and Applications, First Int'l Workshop, Selected Papers,* R. Kasturi and K. Tombre, eds., pp. 270-285, *Lecture Notes in Computer Science,* vol. 1072. Berlin: Springer-Verlag, 1996.

[2] O. Hori and S. Doermann, "Quantitative Measurement of the Performance of Raster-to-Vector Conversion Algorithms," *Graphics Recognition: Methods and Applications, First Int'l Workshop, Selected Papers,* R. Kasturi and K. Tombre, eds., pp. 57-68, *Lecture Notes in Computer Science,* vol. 1072. Berlin: Springer-Verlag, 1996.

[3] L. Wenyin and D. Dori, "A Protocol for Performance Evaluation of Line Detection Algorithms," *Machine Vision and Applications,* vol. 9, nos. 5/6, pp. 240-250, 1997.

[4] L. Wenyin and D. Dori, "A Proposed Scheme for Performance Evaluation of Graphics/Text Separation Algorithms," K. Tombre and A. Chhabra, eds., *Graphics Recognition: Algorithms and Systems, Second Int'l Workshop, GREC '97, Selected Papers,* pp. 359-371, *Lecture Notes in Computer Science,* vol. 1389. Berlin: Springer-Verlag, 1998.

[5] R. Kasturi and I. Phillips, "The First International Graphics Recognition Contest—Dashed-Line Recognition Competition," *Graphics Recognition: Methods and Applications, First Int'l Workshop, Selected Papers,* R. Kasturi and K. Tombre, eds., *Lecture Notes in Computer Science,* vol. 1072. Berlin: Springer-Verlag, 1996.

[6] A. Chhabra and I. Phillips, "Web Page for the Second International Graphics Recognition Contest—Raster to Vector Conversion," http://graphics. basit. com/iapr-tc10/contests/contest97/.

[7] I/Vector (Vectory) ver. 3.8 Raster to Vector Conversion Software, Graphikon, Berlin, Germany, and IDEAL Scanners & Systems, Rockville, Md., http://www. graphikon.com and http://www. ideal.com.

[8] VPstudio ver. 6 rev. 2 Raster to Vector Conversion Software, Softelec, Munich, Germany, and Austin, Tex., http://www. softelec.com and http://www. hybridcad.com.

[9] L. Wenyin and D. Dori, "Automated CAD Conversion with the Machine Drawing Understanding System," *Proc. IAPR Workshop Document Analysis Systems,* pp. 241-259, Malvern, Pa., Oct. 1996, ftp://ftp. technion.ac. il/pub/supported/ie/dori/MDUS/sunmdus.gz.

[10] ECVNet, "Benchmarking and Performance Evaluation Web Site," http://pandora.imag.fr/ECVNet/benchmarking.html.

[11] R. Haralick, "Performance Characterization in Image Analysis: Thinning, a Case in Point," *Pattern Recognition Letters,* vol. 13, pp. 5-12, 1992.

[12] D. Dori, "Object-Process Analysis Maintaining the Balance between System Structure and Behavior," *Logic and Computation,* vol. 5, no. 2, pp. 227-249, 1995.

[13] D. Dori, "Representing Pattern Recognition-Embedded Systems Through Object-Process Diagrams: The Case of the Machine Drawing Understanding System," *Pattern Recognition Letters,* vol. 16, no. 4, pp. 377-384, 1995.

[14] J. O'Rourke, *Computational Geometry in C.* Cambridge, U.K.: Cambridge Univ. Press, 1994. Source code available at ftp://grendel.csc.smith.edu/pub/compgeom.

[15] VOGLE, a public domain device portable graphics library, ftp://munnari.oz.au/pub/graphics/vogle.tar.gz.

[16] S. Leffler and Silicon Graphics, Inc. "TIFF Software Distribution," ftp://ftp. sgi.com/graphics/tiff/tiff-v3.4beta036-tar.gz.

[17] D. Knuth, "The Portable Random Number Generator," http://www-cs-faculty.stanford.edu/knuth/programs.html. Also published in *The Art of Computer Programming, Vol. 2/Seminumerical Algorithms,* third ed., section 3.6. Reading, Mass.: Addison-Wesley, 1997.

[18] A. Chhabra and I. Phillips, "A Benchmark for Graphics Recognition Systems," *Empirical Evaluation Techniques in Computer Vision,* K. Bowyer and P.J. Phillips, eds., pp. 28-44, 1998.

[19] I. Phillips, "Users' Reference Manual," CD-ROM, UW-III Document Image Database-III.

[20] Autodesk, Inc., *AutoCAD Release 13 Customization Guide,* 1995.

[21] H. Baird, "Document Defect Models," *Proc. IAPR Workshop Syntactic and Structural Pattern Recognition (SSPR '90),* pp. 38-46, Murray Hill, N.J., June 1990.

[22] T. Kanungo, R.M. Haralick, and I.T. Phillips, "Nonlinear Local and Global Document Degradation Models," *Int'l J. Imaging Systems and Technology*, vol. 5, no. 4, pp. 220-230, Fall 1994.
[23] T. Kanungo, "Document Degradation Models and a Methodology for Degradation Model Validation," PhD thesis, Univ. of Washington, Seattle, 1996.
[24] A. Chhabra and I. Phillips, "The Second International Graphics Recognition Contest—Raster to Vector Conversion: A Report," *Graphics Recognition: Algorithms and Systems, Second Int'l Workshop, GREC '97, Selected Papers*, pp. 390-410, *Lecture Notes in Computer Science*, vol. 1389. Berlin: Springer-Verlag, 1998.
[25] I. Phillips, J. Liang, A. Chhabra, and R. Haralick, "A Performance Evaluation Protocol for Graphics Recognition Systems," *Graphics Recognition: Algorithms and Systems, Second Int'l Workshop, GREC '97, Selected Papers*, pp. 372-389, *Lecture Notes in Computer Science*, vol. 1389. Berlin: Springer-Verlag, 1998.
[26] D. Dori, "Arc Segmentation in the Machine Drawing Understanding Environment," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 11, pp. 1,057-1,068, Nov. 1995.
[27] D. Dori and L. Wenyin, "Stepwise Recovery of Arc Segmentation in Complex Line Environents," *Int'l J. Document Analysis and Recognition*, vol. 1, no. 1, pp. 62-71, 1998.
[28] J.P. Egan, *Signal Detection Theory and ROC Analysis*. New York: Academic Press, 1975.

Ihsin T. Phillips received the BS degree in 1979, the MS degree in 1981, and the PhD degree in 1984, all in computer science, from the University of Maryland, College Park. In 1984, she joined the Department of Computer Science at the University of Maryland, Baltimore Campus, as an assistant professor. In 1985, she joined the Department of Computer Science and Software Engineering at Seattle University, where she was promoted to associate professor in 1991 and to professor in 1997. Since 1998, she has occupied the Thomas Bannan Endowed Chair in Engineering from the School of Science and Engineering at Seattle University. Since 1989, she has been an affiliate professor of the Department of the Electrical Engineering at the University of Washington and has served on the graduate faculty there since 1992. Her research areas include image processing, pattern recognition, document image understanding, document image database design, and performance evaluation of document image analysis and recognition systems. Her most significant contribution to the field of document image analysis and recognition has been the leadership role she had in the design and creation of the three sets of document image databases: UW-I, UW-II, and UW-III. Since their creation, these three databases have been used by researchers in the field from all over the world for testing and benchmarking their systems. She also helped lead the first (1995), the second (1997), and the third (1999) International Graphic Recognition System Contests on Engineering drawings. She is a member of the IEEE and the IEEE Computer Society.

Atul K. Chhabra (S'85-M'91-SM'99) received the BTech degree in mechanical engineering from the Indian Institute of Technology, Delhi, in 1984 and the PhD degree in electrical engineering from the University of Cincinnati, Cincinnati, Ohio, in 1990. Since 1990, Dr. Chhabra has been working at Bell Atlantic Network Systems, Advanced Technology (formerly known as NYNEX Science & Technology), where he is currently a senior member of technical staff. His research interests include pattern recognition, graphics recognition, document image understanding, computer vision, neural networks, and performance evaluation. For the last several years, he has led the research and development effort in the semi-automatic interpretation of images of hard copy engineering drawings of the telephone company. His past research activities included hand-printed character recognition, forms recognition, and early vision. He helped develop a large document management system geared toward technical drawings. He managed the scanning and semi-automatic indexing of several hundred thousand telephone company drawings. He participated in the NIST contest on OCR systems.

Dr. Chhabra served as a program committee member for several IEEE and IAPR conferences and workshops. He co-chaired the second International Graphics Recognition Workshop (GREC '97) which was held in Nancy, France, in 1997. He is co-chair of the third workshop in this series (GREC '99), which is to be held in Jaipur, India. He co-organized the second International Graphics Recognition Contest at the GREC '99 workshop. He is co-organizing the third contest at GREC '99. Dr. Chhabra is currently the chairman of the IAPR technical committee on graphics recognition. He is a senior member of the IEEE and a member of the IEEE Computer Society and IAPR.