IJDAR (2004) 6: 219–229
Digital Object Identifier (DOI) 10.1007/s10032-003-0106-z

International Journal on Document Analysis and Recognition IJDAR

A fast technique for comparing graph representations with applications to performance evaluation

D. Lopresti[1], **G. Wilfong**[2]

[1] Department of Computer Science and Engineering, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015, USA
[2] Bell Labs, Lucent Technologies Inc., 600 Mountain Avenue, Murray Hill, NJ 07974, USA

Received: 1 October 2002 / Accepted: 15 January 2003
Published online: 6 February 2004 – © Springer-Verlag 2004

Abstract. Finding efficient, effective ways to compare graphs arising from recognition processes with their corresponding ground-truth graphs is an important step toward more rigorous performance evaluation.

In this paper, we examine in detail the graph probing paradigm we first put forth in the context of our work on table understanding and later extended to HTML-coded Web pages. We present a formalism showing that graph probing provides a lower bound on the true edit distance between two graphs. From an empirical standpoint, the results of two simulation studies and an experiment using scanned pages show that graph probing correlates well with the latter measure. Moreover, our technique is very fast; graphs with tens or hundreds of thousands of vertices can be compared in mere seconds. Ease of implementation, scalability, and speed of execution make graph probing an attractive alternative for graph comparison.

Keywords: Graph comparison – Edit distance – Performance evaluation – Document recognition – Layout analysis

1 Introduction

Graphs arise as a fundamental representation throughout much of computer science and engineering, including the field of document analysis. Finding efficient, effective ways to compare graphs is important to areas ranging from basic pattern recognition to sophisticated models for information retrieval.

One potential application lies in the area of performance evaluation, which is crucial to the development of robust document analysis systems. Unfortunately, it is not unusual for researchers to treat this step as somewhat of an afterthought, placing more emphasis on the proposal of new recognition techniques. As a result, evaluation methods often are ad hoc (e.g., manually counting the errors that seem to have arisen or, worse, presenting a few examples for the reader of a paper to inspect visually), employ measures that are specialized to one particular application, or are of mostly theoretical interest and not computationally feasible for realistically sized problem instances.

While it is difficult to draw general conclusions about a field that is so diverse, one point of commonality is the frequent use of graph-oriented data structures to hold the intermediate and/or final results of recognition processes. For example, algorithms for higher-level document understanding tasks often use graphs to encode the logical structure of a page. Given the graph that is output by a document analysis algorithm and its corresponding ground-truth graph, having a measure for comparing them would give us a mechanism for evaluating the performance of the algorithm in question. Because most problems relating to graph comparison have no known solution that is both efficient and guaranteed to be optimal, researchers have devised a wide range of heuristics.

Recently, we have begun to explore an intuitive, easy-to-implement scheme for the problem of performance evaluation when document recognition results are represented in the form of a graph. As shown in Fig. 1, *graph probing* places each of the two graphs under study inside a "black box" capable of evaluating a set of graph-oriented operations (e.g., counting the number of vertices labeled in a certain way) and then poses a series of simple queries derived from the graphs themselves. A measure of their dissimilarity is the degree to which their responses to the probes disagree.

In this paper, we examine in detail the graph probing paradigm we first put forth in the context of our work on table understanding [9–11], where it played an important role in evaluating the performance of the recognition techniques under development. We later extended the approach to the analysis of HTML-coded Web pages from the perspective of information retrieval [16,17]. The preliminary experimental results reported in a short confer-

Correspondence to: D. Lopresti
(e-mail: lopresti@cse.lehigh.edu)

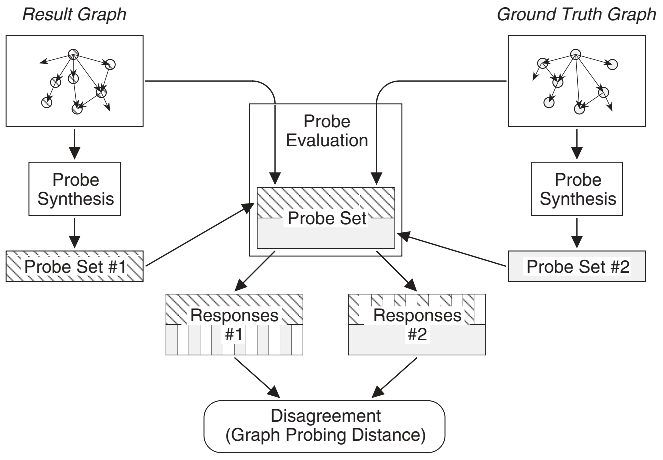

Fig. 1. Overview of graph probing

ence paper [18] are substantially augmented herein and accompanied by more extensive explanations and new analyses.

We begin with a discussion of past work relating to the problem of graph comparison. In Sect. 3, we present a formalism showing that graph probing provides a lower bound on the true edit distance between two graphs. To see how well the approach might work in practice, we examine the results from two simulation studies in Sect. 4, one employing a graph model for hierarchical page layout and the other for the logical structure of tables. We also describe an experiment using scanned page images drawn from a standard dataset to demonstrate the feasibility of applying graph probing to real recognition algorithms. An empirical timing analysis shows that the procedure is extremely efficient; graphs with a million vertices can be compared in less than 10 s. Finally, we offer our conclusions and topics for future research in Sect. 5.

2 Related work

Graph comparison is a widespread yet challenging problem, so it should come as no surprise that many researchers have proposed heuristics and/or solutions designed for special cases. It is not our intent to survey the field exhaustively, but rather to identify certain representative papers, especially those most closely related to the approach we are about to describe. A more comprehensive overview can be found in a recent paper by Bunke [3]. Jolion offers opinions on research trends in graph matching [12].

To begin with, it is important first to distinguish between the exact and approximate matching problems. The former is typically called graph *isomorphism*, while the latter is often phrased in terms of a minimum-cost sequence of basic editing operations (e.g., the insertion and deletion of vertices and edges) that accounts for the observed differences between the two graphs and that defines the notion of *edit distance*. These viewpoints are in fact complementary; it should be clear how a solution to the approximate matching problem could be helpful in solving the isomorphism problem. Moreover, since graphs that are sufficiently similar most likely contain subgraphs that are identical, subgraph isomorphism can be seen as facilitating approximate matching. A formal connection between these two concepts was established by Bunke [2].

In the context of graph editing, another vital distinction arises with respect to two particular quantities that may be of interest: the actual sequence of operations needed to edit one graph into the other and the cost of such a sequence. The former is useful in attempting to understand the differences between the graphs and why they may have arisen, while the latter provides a concrete measure of similarity. Given a minimum-cost sequence of editing operations, calculating the corresponding edit distance is straightforward. While the converse is not true, edit distance by itself is still extremely valuable, especially if it can be computed much more rapidly or for larger graphs than would otherwise be possible using procedures that return the operations.

Much prior work has focused on the graph isomorphism problem (i.e., finding an exact correspondence between two graphs) and its variants. The complexity of graph isomorphism remains unresolved, and, unfortunately, all known algorithms for its solution have worst-case exponential running times [7]. Heuristics for determining isomorphism often rely on the concept of a *vertex invariant*, that is, a value $f(v)$ assigned to each vertex v such that under any isomorphism I, if $I(v) = v'$, then $f(v) = f(v')$. One such invariant is the degree of a vertex (or the in- and out-degrees, if the graph is directed). Indeed, **nauty**, an effective software package for computing graph isomorphism ([19,20]), relies on vertex invariants.

In general, such heuristics can fail in a catastrophic manner [5]. On the other hand, it has been shown that for random graphs, there is a simple linear time test for checking if two graphs are isomorphic based on the degrees of the vertices, and this test succeeds with high probability [1].

Other research aims at speeding up the computation for database searches. Lazarescu et al. propose a machine learning approach to building decision trees for eliminating from further consideration graphs that cannot possibly be isomorphic to a given query graph [15]. While they employ a similar set of features to the ones we use, they do not consider the approximate matching problem. Bunke and Messmer present a decision-tree-based precomputation scheme for solving the subgraph isomorphism problem, although their data structure can be exponential in the size of the input graphs in the worst case [4,21].

Valiente and Martínez describe an approach to subgraph pattern matching based on finding homomorphic images of every connected component in the query [30]. Again, the worst-case time complexity is exponential, but such features could also perhaps be incorporated in the measure we are about to present.

Turning to graph edit distance, we note there have been a large number of papers written on the subject and its many applications. These can be divided into two categories. In the first, we have procedures that are guaranteed to find an optimal solution but may, in the worst case, require exponential time, such as the early work by Fu and colleagues [27,29]. The second category includes heuristics that may not necessarily return an optimal match but that have polynomial running times as in, for example, the Bayesian framework posed by Myers et al. [22]. Frequently these papers focus on search strategies intended to speed up the computation when certain conditions are satisfied, making the understanding and implementation of the algorithms more difficult.

Papadopoulos and Manolopoulos discuss an idea that is philosophically quite similar to the one we are exploring [25]. However, they focus on a single invariant: vertex degree. It is clear this is not sufficient for catching all of the interesting differences that can arise among document representations. We would like to determine a range of possible graph features that can be used to distinguish the sorts of effects seen in practice.

Instead of trying to solve the problem for graphs in general, some leeway can be had by limiting the discussion to trees, for which efficient comparison algorithms are known. Schlieder and Naumann consider a problem closely related to ours: error-tolerant embedding of trees to judge the similarity of XML documents [28]. Likewise, Dubois et al. write about tree embedding for searching databases of semistructured multimedia documents and for query-by-example [6].

3 A formalism for graph probing

In this section, we formalize the concept of graph probing as a way of quantifying graph similarity. Our goal is to relate probing to more rigorous but harder-to-compute graph edit distance models.

While ultimately we are interested in a more general class of graphs, to begin with let $G_1^u = (V_1, E_1)$ and $G_2^u = (V_2, E_2)$ be two undirected graphs. Consider a graph editing model that allows the following basic operations: (1) delete an edge, (2) insert an edge, (3) delete an isolated vertex, (4) insert an isolated vertex. It should be clear that such operations can be used to edit any graph into any other graph. The minimum number of operations needed to edit G_1^u into G_2^u is the undirected graph edit distance, $dist^u(G_1^u, G_2^u)$. There is no known algorithm for efficiently computing this distance in general.

Now consider a probing procedure that, for a specific vertex degree n, asks the following question: "How many vertices with degree n are present in graph $G^u = (V, E)$?" Let PR_{1a} collect the responses for all vertex degrees represented in the graph (the response for all other vertex degrees is, of course, implicitly zero):

$$PR_{1a}(G^u) \equiv (n_0,\ n_1,\ n_2,\ \ldots) \qquad (1)$$
$$\text{where } n_i = |\{v \in V \mid deg(v) = i\}|$$

Then define $probe^u(G_1^u, G_2^u) \equiv PR_{1a}(G_1^u) - PR_{1a}(G_2^u)$ as the L_1 norm of the two vectors; that is, $probe^u$ is the magnitude of the difference between the two sets of probing results.

Theorem 1. *Under the undirected graph model and its associated edit model, $probe^u$ is a lower bound, within a factor of four, on the true edit distance between any two graphs. That is, $probe^u(G_1^u, G_2^u) \leq 4 \cdot dist^u(G_1^u, G_2^u)$.*

Sketch of Proof: The proof of this theorem follows from a simple case analysis. The operations that cause the largest possible disparity between edit distance and the graph probing measure are the deletion or insertion of an edge. In particular, consider a pair of vertices, v_1 and v_2, where v_1 has degree δ_1 and v_2 has degree δ_2. Now add an edge between them. Since the degree of v_1 increases from δ_1 to $\delta_1 + 1$ and the degree of v_2 increases from δ_2 to $\delta_2 + 1$, this edit can result in there being one more vertex with degree $\delta_1 + 1$ and one more with degree $\delta_2 + 1$. It can also result in there being one fewer vertex with degree δ_1 and one fewer with degree δ_2. Thus, this one operation may cause as many as, but no more than, four of the probes to differ by one. ■

An example is illustrated in Fig. 2a, where the comparison of the two probe vectors PR_{1a} yields a value of four as opposed to the true edit distance, which is one (corresponding to the insertion of a single edge). This demonstrates the tightness of the lower bound.

The time needed to perform the above probing procedure is $O(\max(|V_1|, |E_1|, |V_2|, |E_2|))$. In the offline case, we can precompute the probes and their responses for one of the graphs. While the worst-case time complexity remains unchanged, the precomputation and an efficient coding of its output can yield a substantial savings.

Unfortunately, there is no guarantee that the above bound is always tight. Indeed, it is not even as tight as the bound that can be derived for the measure described in [25]. It does appear to be easier to generalize, however, and the importance of this will become apparent shortly.

To proceed to the case of directed graphs, we can consider the same set of editing operations (recognizing that the edges are now directed) and change the probes to be: "How many vertices with in-degree m and out-degree n are present in graph G^d?" As before, let PR_{1b} collect the responses for all vertex in- and out-degrees present in the graph,

$$PR_{1b}(G^d) \equiv (n_{0,0},\ n_{0,1},\ n_{1,0},\ \ldots) \qquad (2)$$
$$\text{where } n_{i,j} = |\{v \in V \mid indeg(v) = i,\ outdeg(v) = j\}|$$

Then define $probe^d(G_1^d, G_2^d) \equiv PR_{1b}(G_1^d) - PR_{1b}(G_2^d)$, leading to an analogous result:

Theorem 2. *Under the directed graph model and its associated edit model, $probe^d$ is a lower bound, within a factor of four, on the true edit distance between any two graphs. That is, $probe^d(G_1^d, G_2^d) \leq 4 \cdot dist^d(G_1^d, G_2^d)$.*

Moreover, $probe^u(G_1^d, G_2^d) \leq probe^d(G_1^d, G_2^d)$.

As before, an example is shown in Fig. 2b. Note that in this case $probe^d$ is identical to the edit distance.

The same observations made above concerning computation time apply here as well. As stated in the theorem, the bound returned by the new, more specific class

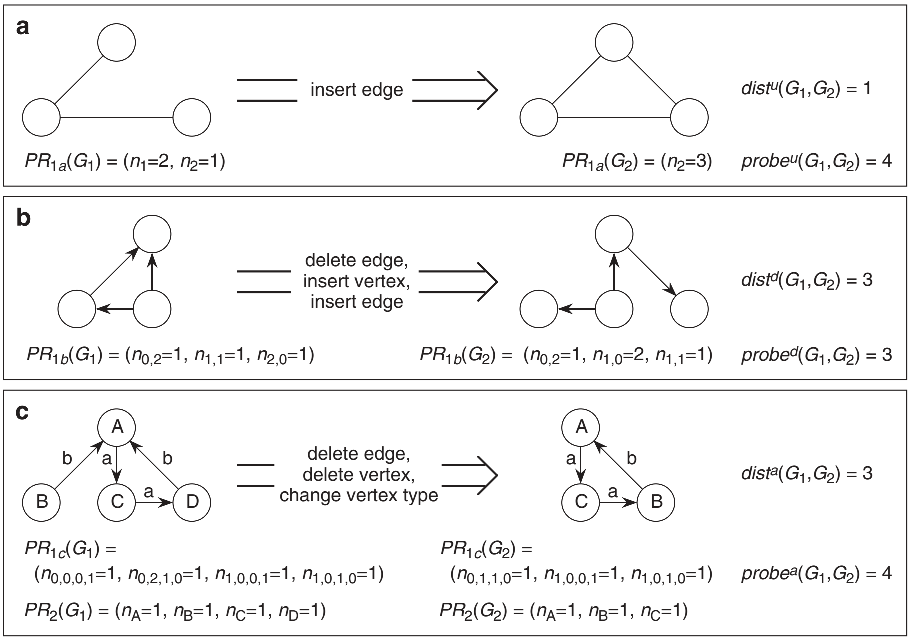
a
insert edge
$dist^u(G_1,G_2) = 1$
$PR_{1a}(G_1) = (n_1=2,\ n_2=1)$
$PR_{1a}(G_2) = (n_2=3)$
$probe^u(G_1,G_2) = 4$
b
delete edge, insert vertex, insert edge
$dist^d(G_1,G_2) = 3$
$PR_{1b}(G_1) = (n_{0,2}=1,\ n_{1,1}=1,\ n_{2,0}=1)$
$PR_{1b}(G_2) = (n_{0,2}=1,\ n_{1,0}=2,\ n_{1,1}=1)$
$probe^d(G_1,G_2) = 3$
c
A B C D a b
delete edge, delete vertex, change vertex type
$dist^a(G_1,G_2) = 3$
$PR_{1c}(G_1) = (n_{0,0,0,1}=1,\ n_{0,2,1,0}=1,\ n_{1,0,0,1}=1,\ n_{1,0,1,0}=1)$
$PR_{1c}(G_2) = (n_{0,1,1,0}=1,\ n_{1,0,0,1}=1,\ n_{1,0,1,0}=1)$
$probe^a(G_1,G_2) = 4$
$PR_2(G_1) = (n_A=1,\ n_B=1,\ n_C=1,\ n_D=1)$
$PR_2(G_2) = (n_A=1,\ n_B=1,\ n_C=1)$

Fig. 2a–c. Probing examples for the three graph models

of probes is at least as good as the original class and sometimes better: $probe^u(G_1^d, G_2^d) \le probe^d(G_1^d, G_2^d)$.[1]

Now generalize the graph model further so that vertices and edges are potentially labeled by a type. For example, vertices might be labeled as corresponding to logical structures within a document (e.g., zone, line, word) along with any associated content, while edges are labeled to represent relationships between structures (e.g., contains, next). To handle such attribute graphs, the edit model previously defined must be expanded to include additional operations: (5) change the type of an edge, (6) change the type of a vertex. The edges and vertices created through insertion operations can be assigned any type initially.

In terms of probing, note that now two graphs can be different (in terms of the types of their vertices and edges) and yet appear to be structurally identical. To deal with this, the probes for counting in- and out-degrees are made specific to edge type. Suppose there are α different edge labels $l_1, \ldots, l_\alpha$. The edge structure of a given vertex can then be represented as a 2α-tuple of nonnegative integers, $(x_1, \ldots, x_\alpha, y_1, \ldots, y_\alpha)$, if the vertex has exactly x_i incoming edges labeled l_i and exactly y_j outgoing edges labeled l_j for $1 \le i, j \le \alpha$. Then a typical probe will have the form: "How many vertices with edge structure $(x_1, \ldots, x_\alpha, y_1, \ldots, y_\alpha)$ are present in graph G^a?" We also need to add a new class of probes focusing on just the vertices and their types: "How many vertices labeled *vtype* are present in graph G^a?" Let PR_{1c} collect the responses for vertex in- and out-degrees and their respective edge types, and let PR_2 collect the responses for vertex types.

[1] The function $probe^u$ is applied to directed graphs G_1^d and G_2^d by simply ignoring the directions on the edges.

Define

$$probe^a(G_1^a, G_2^a) \equiv \qquad (3)$$
$$(PR_{1c}(G_1^a) - PR_{1c}(G_2^a)) + (PR_2(G_1^a) - PR_2(G_2^a))$$

We then have:

Theorem 3. *Under the attribute graph model and its associated edit model, $probe^a$ is a lower bound, within a factor of four, on the true edit distance between any two graphs. That is, $probe^a(G_1^a, G_2^a) \le 4 \cdot dist^a(G_1^a, G_2^a)$.*
Moreover,

$$probe^u(G_1^a, G_2^a) \le probe^d(G_1^a, G_2^a) \le probe^a(G_1^a, G_2^a)$$

This is depicted in Fig. 2c.

The precomputation needed for each graph is as follows. Computing the edge structures of all the vertices takes total time $O(|E| + \alpha|V|)$. These $|V|$ 2α-tuples can then be lexicographically sorted in $O(\alpha(\delta + |V|))$ time, where δ is the maximum number of edges incident on any vertex. Then a simple pass through the sorted list allows us to compute the number of vertices in each of the (nonempty) classes in additional time $O(\alpha|V|)$. Thus the total precomputation time is $O(\alpha(\delta + |V|) + |E|)$. Since α and δ are likely to be small constants, the time is essentially the same as for the case of undirected graphs.

We refer to the quantity $probe^a(G_1^a, G_2^a)$ defined by Eq. 3 as the *graph probing distance* between G_1^a and G_2^a. It provides an approximation to the true edit distance between two graphs, which would be too expensive to compute in the most general case. Table 1 summarizes the results of this section.

Before concluding this portion of the presentation, we note that the various graph probing measures satisfy the triangle inequality. In other words, $probe(G_1, G_2) \ge probe(G_1, G_3) + probe(G_3, G_2)$ for all graphs G_1, G_2, and G_3 of the appropriate class. Graph probing distance is

Table 1. Summary of the various models

Graph model	Edit model	Probe model	Bound
Undirected	(1) Delete edge (2) Insert edge (3) Delete vertex (4) Insert vertex	(a) Vertex degree	$probe^u(G_1^u, G_2^u) \leq 4 \cdot dist^u(G_1^u, G_2^u)$
Directed	As above	(a) Vertex in- and out-degree	$probe^d(G_1^d, G_2^d) \leq 4 \cdot dist^d(G_1^d, G_2^d)$
Attribute	As above, plus (5) Change edge type (6) Change vertex type	(a) Vertex in- and out-degree by edge type (b) Vertex type	$probe^a(G_1^a, G_2^a) \leq 4 \cdot dist^a(G_1^a, G_2^a)$

also nonnegative and symmetric (i.e., $probe(G_1, G_2) = probe(G_2, G_1)$), thus satisfying three of the four conditions of a metric space. The remaining condition, separation, is violated, however, as $probe(G_1, G_2) = 0$ does not imply that G_1 and G_2 are identical (isomorphic). Hence, graph probing distance forms what is commonly referred to as a "pseudometric" space.

4 Experimental results

Based on the preceding formalism, two probe classes are applicable to the kinds of attribute graphs that arise in document analysis applications:

Class 1c These probes examine the vertex and edge structure of the graph by counting in- and out-degrees, tabulating different types of incoming and outgoing edges separately. An example is: *How many vertices with one incoming edge labeled "contains" and another labeled "next," along with two outgoing edges labeled "contains" and one labeled "next," are present in the graph?*

Class 2 These probes count the occurrences of a given type of vertex in the graph. Recall that "type" refers both to a logical label as well as any content that may be associated with the vertex. A representative Class 2 probe might be paraphrased as: *How many vertices labeled "Word" with content "pentagon" are present in the graph?*

For implementing such a protocol, we have developed a graph probing library as an extension to a general-purpose programming language, Tcl/Tk [24].

To test these ideas, we designed a series of simulation studies as well as an experiment using results from a well-known page segmentation algorithm. As previously noted, we would like to be able to equate graph probing distance (i.e., Eq. 3) with some formal notion of correspondence. Unfortunately, there is no measure that is both universal and easy to compute that we can use for comparison purposes (indeed, this point is a primary motivation of our research). Hence, we have chosen to work "backwards" by randomly generating a ground-truth graph and then simulating recognition "errors" by editing the graph in various ways: adding and deleting vertices, altering labels and content, etc. The number of edits we perform is an approximation (an upper bound, in fact) of the actual distance between two graphs.

4.1 Entity graph model simulation

The *entity graph* model reflects a standard document hierarchy: vertices labeled as *Page*, *Zone*, *Line*, or *Word* (see, e.g., [13]). The edge structure represents two relationships: *contains* and *next*. A small example of such an entity graph is shown in Fig. 3, corresponding to the nonsense document fragment given below:

```
satisfactory extrinsic inexpert frankfurter

abutting tarantula
grillwork pentagon attribution bilharziasis
```

The random entity graphs used in our study are, on average, eight times larger than this.

We begin by creating a graph for a page with a random number of zones. For each zone, we then generate a random number of lines, and for each line a random number of words. Content for *Word* vertices is chosen to be a word randomly selected from the Unix **spell** dictionary. The editing operations used to simulate recognition errors are guaranteed to yield another legal entity graph. These include altering the content of a *Word* vertex, deleting an existing *Word*, *Line*, or *Zone* vertex (and its associated edges and children), or inserting a new *Word*, *Line*, or *Zone* vertex along with the corresponding content (if any) that lies below it in the hierarchy.

The entire simulation involved generating a total of 2500 "ground-truth" entity graphs, performing a randomly selected number of edits on each, synthesizing and evaluating Class 1c and 2 probes, and gathering relevant statistics. The results for this experiment are presented in Table 2 and Fig. 4. As can be seen from the upper table, there was a wide range in the size of the graphs under consideration. On average, approximately one probe was generated per vertex. Overall, the average graph probing distance was 26.3, and the minimum was 2 (i.e., the

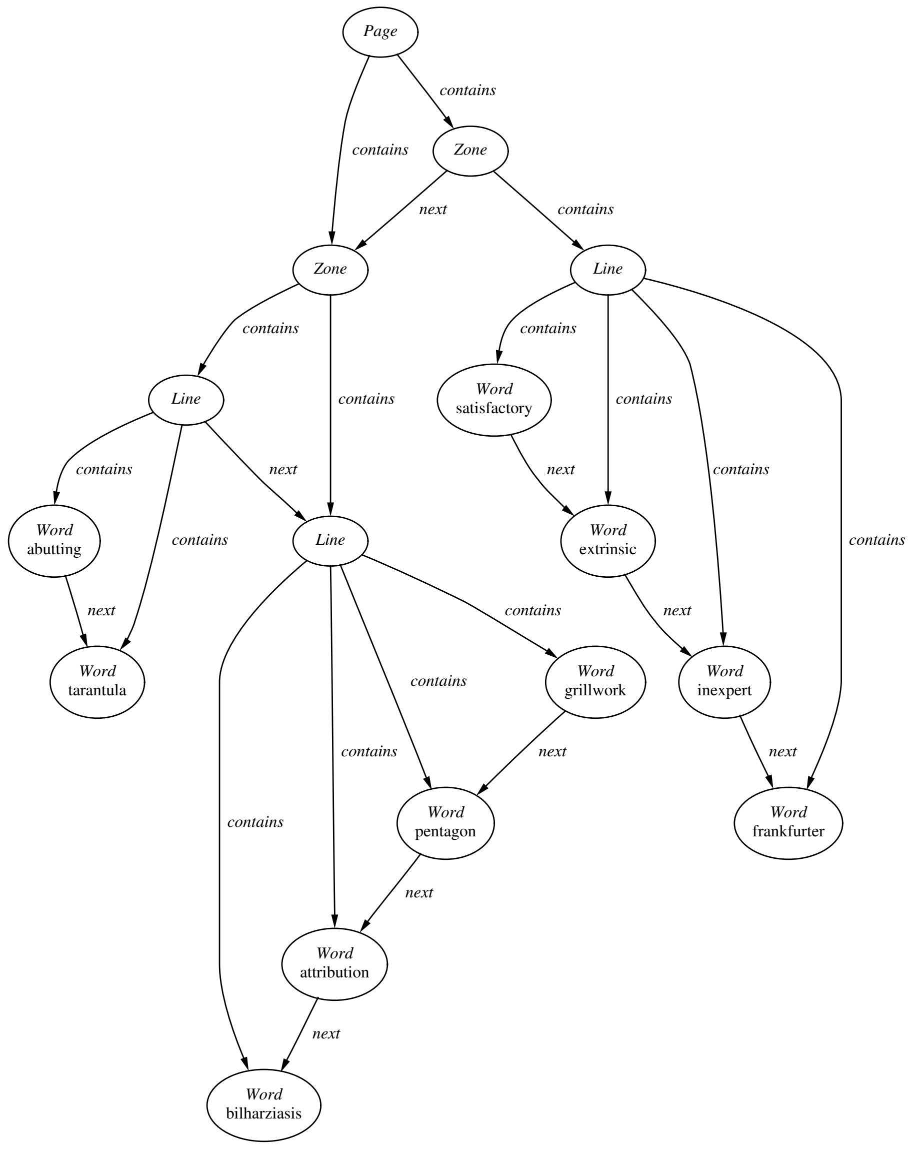

Fig. 3. An instance of an entity graph

probes always captured the fact that one of the graphs contained errors).

The ability of the two probe classes to differentiate the two graphs is shown in the lower part of Table 2. Class 2 probes never failed in this experiment. Note that Class 1c probes will always miss differences that involve only content, but offsetting edits have the potential to confuse either of the classes. The last column in this table indicates that there were 91 graph-pairs that were distinguished only by using Class 2 probes.

The number of graph editing operations as a function of graph probing distance is displayed in Fig. 4. The datapoints show the average at each step along the x-axis, while the vertical bars give the min/max range. Also displayed are the theoretical lower bound and the least squares error fit to the data points; the average error in the latter case is 4.41. Hence, the graph probing distance (i.e., Eq. 3) provides a reasonably dependable measure of the difference between two graphs.

4.2 Table graph model simulation

Entity graphs encode document page structure in a very general way. A more restricted type of graph is the *table graph*, as defined in our past work on table recognition [9–11]. Here we employ a slightly simplified version of that model. Tables consist of lower-level cells grouped in terms of logical rows and columns. Hence, vertices in table graphs can be labeled *Cell*, *Row*, *Column*, and, ultimately, *Table*. Edges encode the *contains* relationship. An example of a table graph is shown in Fig. 5, as derived from the following randomly generated table with four rows and three columns:

```
regression radiant    474383991  sima Nostrand
   clubroom incuse    593134723     ant Sussex
       ascribe gam   1813217419        opulent
shovel registrable    615003753   astride Peru
```

Along the same lines as the previous simulation, we begin by generating a ground-truth graph containing a random number of rows and columns. Each column is

Table 2. Statistics for the entity graph simulation (2500 random graphs)

Attribute	Min	Max	Avg
Zones	1	8	4.4
Lines	1	53	19.8
Words	2	277	98.8
Total vertices	5	337	124.0
Edits	1	178	23.4
Class 1c probes	10	52	34.2
Class 1c distance	0	121	16.3
Class 2 probes	10	555	203.4
Class 2 distance	1	72	10.1
Overall probes	20	595	237.5
Overall distance	2	193	26.3
Probes/vertex	0.869	2.000	1.014

Probes	Detected	Missed	Rate	Unique
Class 1c	2,409	91	96.4%	0
Class 2	2,500	0	100.0%	91
Overall	2,500	0	100.0%	n/a

Table 3. Statistics for the table graph simulation (2500 random graphs)

Attribute	Min	Max	Avg
Rows	4	20	12.1
Columns	4	8	6.0
Total vertices	25	189	91.8
Edits	1	79	25.5
Class 1c probes	6	8	7.9
Class 1c distance	0	102	29.7
Class 2 probes	38	326	151.2
Class 2 distance	2	46	13.7
Overall probes	44	334	159.1
Overall distance	2	132	43.4
Probes/vertex	0.828	0.900	0.867

Probes	Detected	Missed	Rate	Unique
Class 1c	2,112	388	84.5%	0
Class 2	2,500	0	100.0%	388
Overall	2,500	0	100.0%	n/a

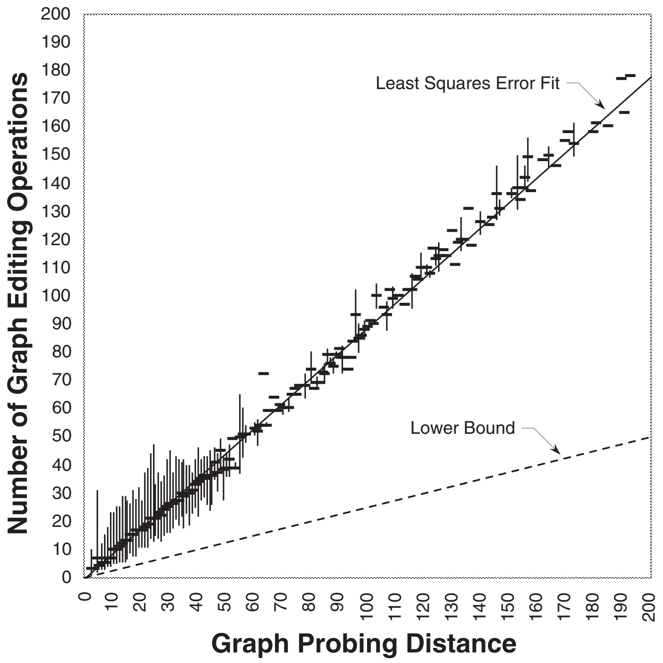

Fig. 4. Results for the entity graph simulation (2500 random graphs)

randomly designated as being either alphabetic or numeric. For the former, table cells are selected to be a string of one or more words, while for the latter the contents of cells are assigned to be random integers. Cells in the first column are always set to be alphabetic (to represent row headers). Editing operations include changing the contents of a *Cell* vertex or deleting or inserting a *Row* or *Column*.

Table 3 and Fig. 6 present the results of running this simulation for 2500 random tables. These graphs were somewhat smaller than those for the entity graph experiment. As Table 3 indicates, the Class 2 probes never failed, but the Class 1c probes missed 388 of the edited tables. This is probably because they compare vertex in- and out-degrees, and any two tables with the same numbers of rows and columns will appear to be identical in this regard. Errors that involve only content will create such confusion, as can offsetting edits. For example, deleting one row and inserting a new row at another location will result in a table with the same vertex and edge structure. Probes from Class 2 will catch this difference, but those from Class 1c cannot. Overall, the average graph probing distance was found to be 43.4.

The number of editing operations as a function of graph probing distance is charted in Fig. 6, along with the lower bound and the least squares error fit (with average error 5.61). As before, this value appears to be a good predictor of the number of edits used to simulate recognition errors, although the behavior of graph-pairs near the extremes of the ranges merits closer examination. Recall that the count of editing operations displayed along the y-axis is only an upper bound on the actual edit distance. It seems possible that worst-case scenarios in the random editing process, e.g., altering the content of several cells in a given row and then later choosing to delete the whole row, could make the edit distance appear substantially larger than it is in reality. This may explain some of the wide deviations for datapoints on the left in the chart.

Since the equation for the least squares error fit line in Fig. 6 is clearly different from that for the line in Fig. 4 (in particular, the slope is noticeably shallower), it appears likely the mapping from graph probing distance to edit distance will need to be calibrated on a per-model basis. This is a subject for future research.

In previous papers [9–11,18], we have also considered a third, more sophisticated class of probes for the table graph model:

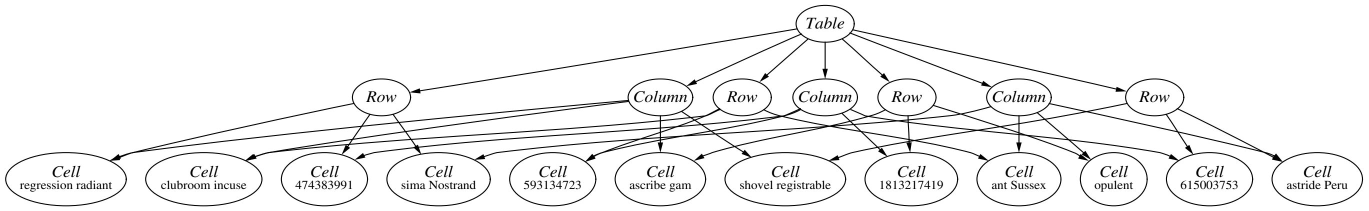

Fig. 5. An instance of a table graph

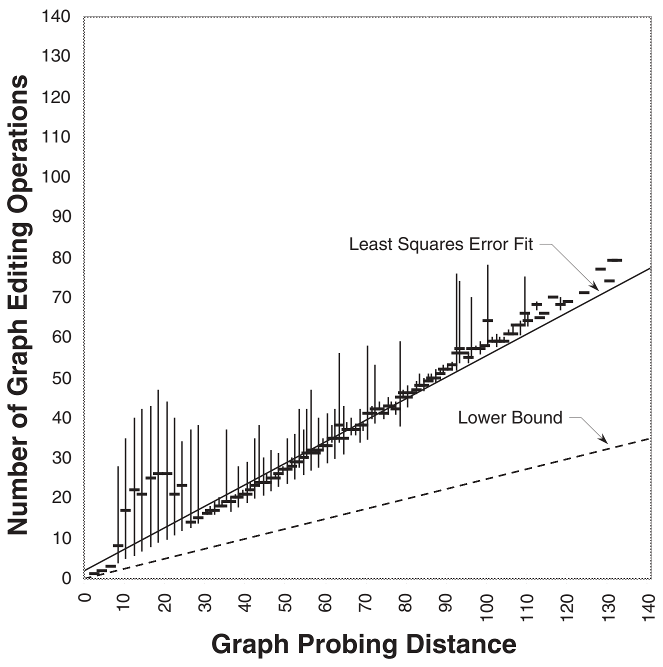

Fig. 6. Results of the table graph simulation (2500 random graphs)

Class 3 For a given target vertex, keys that uniquely determine its row and column are identified. These are used to index into the graph, retrieving the content of the vertex that lies at their intersection. Again referring to the table shown in Fig. 5, an example might be: *What is the content of the cell that lies at the intersection of the row indexed by "clubroom incuse" and the column indexed by "sima Nostrand"?*

Our intention was that such probes mimic the sorts of operations users may wish to perform on the results of document analysis (e.g., table lookups in the style of relational database queries) and hence more accurately reflect the impact of recognition errors on a real application.

While this may indeed be the case, we have not yet found a way to relate other classes of probes to the formalism we presented earlier in Sect. 3. Still, it is worth examining the potential utility of such comparison measures, so we performed another, separate simulation study involving 500 randomly generated graphs following the same procedure as before, only this time comparing the ground-truth and edited graphs using Class 3 probes. These results are presented in Table 4 and Fig. 7.

Table 4. Statistics for the table graph simulation using Class 3 probes (500 random graphs)

Attribute	Min	Max	Avg
Rows	4	20	12.0
Columns	4	8	6.0
Total vertices	25	189	91.1
Edits	1	81	26.1
Class 3 probes	36	320	144.3
Class 3 distance	3	118	27.4
Overall probes	36	320	144.3
Overall distance	3	118	27.4
Probes/vertex	0.655	0.847	0.774

Probes	Detected	Missed	Rate	Unique
Class 3	500	0	100.0%	n/a
Overall	500	0	100.0%	n/a

As can be seen, the statistics are for the most part comparable to the previous table experiment. The Class 3 probes are far more expensive to evaluate, however. Although they always correctly detected that there was a difference between the graphs, the plot in Fig. 7 suggests that these probes are not necessarily well correlated with the notion of graph edit distance, so one is probably not a good substitute for the other. Nevertheless, Class 3 probes still seem to present some attractive qualities that merit further investigation.

4.3 Page segmentation experiment

Our past experience using graph probing for performance evaluation in small-scale experiments involving real (as opposed to simulated) document analysis results has been quite favorable (see [9–11]). As is often the case, however, the considerable effort required to create the necessary ground-truth presents a barrier to performing larger studies featuring our table understanding work at the present time (for a discussion of some of the associated issues, see [8]). Instead, we developed a test using a well-known page segmentation technique, Nagy and Seth's X-Y cut algorithm [23]. This partitions a page image recursively, representing the result as a tree. By injecting a controlled amount of random "bit-flip" noise (turning black pixels white as might arise in a light photocopy), we can induce differences in the segmentation graph that hopefully will be reflected in the distance computed during probing.

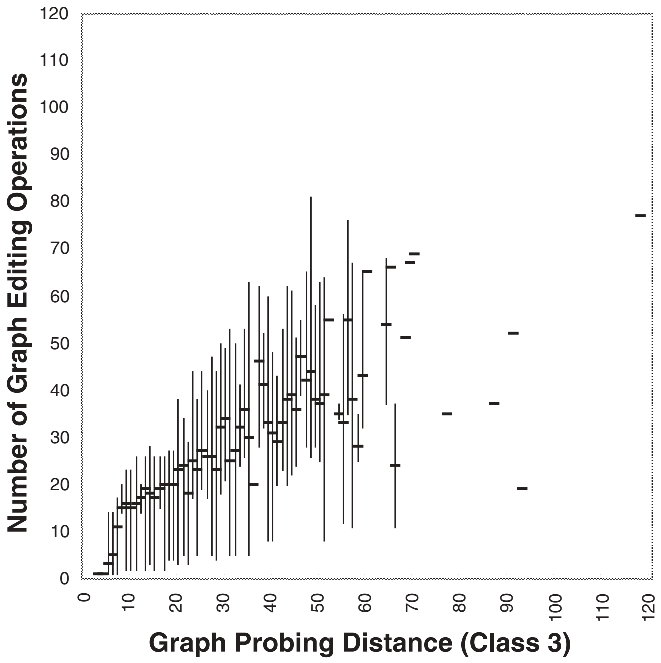

Fig. 7. Results of the table graph simulation using Class 3 probes (500 random graphs)

Table 5. Statistics for the page segmentation experiment (150 page images)

Attribute	Min	Max	Avg
X-Cuts	300	1,097	510.8
Y-Cuts	58	668	210.4
Text	246	987	480.6
Other	0	29	10.2
Total vertices	641	2,501	1,212.0
Class 1c probes	8	8	8.0
Class 1c distance	0	1,186	179.0
Class 2 probes	1,283	4,780	2,424.1
Class 2 distance	41	4,122	1,099.8
Overall probes	1,291	4,788	2,432.1
Overall distance	47	4,566	1,278.8
Probes/vertex	1.002	1.006	1.004

Probes	Detected	Missed	Rate	Unique
Class 1c	145	5	96.7%	0
Class 2	150	0	100.0%	5
Overall	150	0	100.0%	n/a

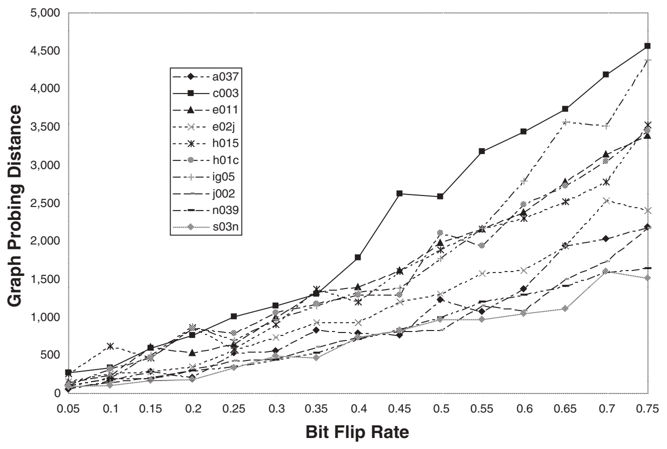

Fig. 8. Results of the page segmentation experiment (150 page images)

The test collection consisted of ten pages taken from the UW1 dataset [26]: a037, c003, e011, e02j, h01c, h015, ig05, j002, n039, and s03n. We chose examples with relatively complex layouts, so that the X-Y cut graphs would be interesting. Each page was subjected to the bit-flip noise at rates ranging from 5% to 75% in increments of 5%. We then compared the segmentation graphs for the original and noisy pages using our graph probing paradigm.

Basic statistics for the probing evaluation of the 150 test pages are presented in Table 5. There were errors in every one of the recognized pages, and this is reflected in the minimum overall distance of 47. As in the simulations, a relatively small number of probes were generated for each vertex in the graphs.

The lower portion of Table 5 shows that, in nearly every case, both probe classes were capable of detecting that there were differences between the segmentation graphs for the original and degraded documents. Only in five instances did the Class 2 probes outperform the other class.

The remaining issue, then, is how well graph probing correlates with the controlled amount of damage we inflicted on each page image. These results are plotted in Fig. 8. Here we show a distinct style of datapoint for each of the ten source documents in the test set. While the correspondence is perhaps coarser than in the simulations, which likewise involved Class 1c and 2 probes, an overall monotonic behavior is still visible.

4.4 Empirical timing analysis

As noted in Sect. 3, graph probing is an efficient procedure; the asymptotic time complexity is essentially $O(|E| + |V|)$ for the kinds of graphs we are interested in. The implementations described in the previous subsections were programmed in Tcl/Tk, which offers a great deal of flexibility but is inefficient from a computational standpoint because it is an interpreted language. To obtain a more accurate estimate of the time needed to perform graph probing, we rewrote the entity graph simulation in C and ran it on an IBM ThinkPad T23 (1 Ghz Pentium III, 256 MB RAM) under Linux, using it to compare random graphs of increasingly larger sizes. The chart in Fig. 9 presents timing results for graphs ranging from 10 000 to 1 000 000 vertices, where the time reported is the total for generating two random graphs of the given size, synthesizing and evaluating the respective

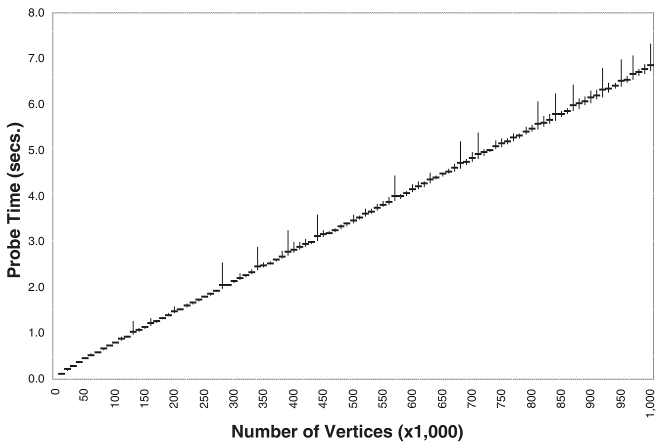

Fig. 9. Timing results of comparing entity graphs of various sizes using graph probing

probe sets, and comparing the output vectors to yield the probing distance. Note that the rise in computation time is strictly linear and that graphs with a million vertices can be compared in less than 10 s.

5 Conclusions

This paper has described a fast, intuitive, easy-to-implement scheme for the problem of performance evaluation when document recognition results are represented in the form of a graph. Graph probing can be seen as having its roots in past work on heuristics for solving the graph isomorphism problem. However, its utility extends beyond simply testing graphs for equivalence; it also allows us to quantify the similarity between two graphs. Building on our past work on table understanding, we provided a formalism proving that graph probing provides a lower bound on the true edit distance. We presented results from two simulation studies using different graph models and an experiment employing real page image data to demonstrate the applicability of the approach. Moreover, our technique is very fast; graphs with tens or hundreds of thousands of vertices can be compared in mere seconds.

There are a number of ways in which this work could be continued. It may be possible to broaden the formalism we presented to cover other error models or probe classes (e.g., Class 3 probes, which mimic simple database queries). The probing paradigm as we have defined it is an offline procedure (i.e., all of the probes are computed in advance, before any are evaluated). Allowing the probing to take place online, making it adaptive, might bring significant benefits.

One of our underlying assumptions throughout this paper has been that all of the basic graph editing operations have the same constant cost. It is conceivable, though, that this could be relaxed to allow more flexible cost functions.

Currently, graph probing provides a measure of how dissimilar two graphs are. It does not, however, produce a mapping from one graph to the other. To identify the potential errors that may have arisen in a recognition process, we must be able to determine a correspondence between the vertices and edges of the graphs as well as associate editing operations with the differences that remain.

Finally, other applications could make use of this technique for graph comparison. In information retrieval, for example, queries and target documents are sometimes represented in terms of graphs. Certain of these cases will probably require retargeting graph probing to address the related subgraph comparison problem (i.e., whether a given graph contains a subgraph that is similar to another graph of interest). See [16,17] for work on comparing semistructured documents that arise from parsing HTML-coded Web pages.

Acknowledgements. Jianying Hu and Ramanujan Kashi had important roles in the table recognition research that led to the development of the graph probing paradigm. The graph drawings shown in this paper were generated using the **dot** tool built by Eleftherios Koutsofios and Stephen North [14]. A preliminary version of the experimental results in this paper was reported at the *Sixth International Conference on Document Analysis and Recognition* [18]. The lower bound proof for graph probing first appeared in a paper at the *Seventh International Workshop on Multimedia Information Systems* [17]. We wish to thank IJDAR editor Karl Tombre and the anonymous referees for their suggestions, which helped improve the paper.

References

1. Babai L, Erdös P, Selkow SM (1980) Random graph isomorphism. SIAM J Comput 9(3):628–635
2. Bunke H (1997) On a relation between graph edit distance and maximum common subgraph. Patt Recog Lett 18:689–694
3. Bunke H (2000) Recent developments in graph matching. In: Proceedings of the 15th International Conference on Pattern Recognition, Barcelona, Spain, September 2000, 2:117–124
4. Bunke H, Messmer BT (1997) Recent advances in graph matching. Int J Patt Recog Artif Intell 11(1):169–203
5. Corneil DG, Kirkpatrick DG (1980) A theoretical analysis of various heuristics for the graph isomorphism problem. SIAM J Comput 9(2):281–297
6. Dubois D, Prade H, Sèdes F (1999) Some uses of fuzzy logic in multimedia databases querying. In: Proceedings of the Workshop on Logical and Uncertainty Models for Information Systems, London, July 1999, pp 46–54
7. Fortin S (1996) The graph isomorphism problem. Department of Computer Science Technical Report TR 96-20, University of Alberta, Canada
8. Hu J, Kashi R, Lopresti D, Nagy, Wilfong G (2001) Why table ground-truthing is hard. In: Proceedings of the 6th International Conference on Document Analysis and Recognition, Seattle, September 2001, pp 129–133
9. Hu J, Kashi R, Lopresti D, Wilfong G (2000) A system for understanding and reformulating tables. In: Proceedings of the 4th IAPR International Workshop on Docu-

ment Analysis Systems, Rio de Janeiro, December 2000, pp 361–372
10. Hu J, Kashi R, Lopresti D, Wilfong G (2001) Table structure recognition and its evaluation. In: Proceedings of Document Recognition and Retrieval VIII, San Jose, January 2001, 4307:44–55
11. Hu J, Kashi R, Lopresti D, Wilfong G (2002) Evaluating the performance of table processing algorithms. Int J Doc Anal Recog 4(3):140–153
12. Jolion JM (2001) Graph matching: what are we really talking about? In: Proceedings of the 3rd IAPR Workshop on Graph-Based Representations in Pattern Recognition, Ischia, Italy, May 2001. http://citeseer.nj.nec.com/503443.html
13. Kanungo T, Lee CH, Czorapinski J, Bella I (2001) TRUEVIZ: a groundtruth / metadata editing and visualizing toolkit for OCR. In: Proceedings of Document Recognition and Retrieval VIII, San Jose, January 2001, 4307:1–12
14. Koutsofios E, North SC (1991) Drawing graphs with dot. Technical Report 59113-910904-08TM, AT&T Bell Laboratories
15. Lazarescu M, Bunke H, Venkatesh S (2000) Graph matching: fast candidate elimination using machine learning techniques. In: Advances in pattern recognition. Lecture Notes in Computer Science, vol 1876, Springer, Berlin Heidelberg New York, pp 236–245
16. Lopresti D, Wilfong G (2001a) Applications of graph probing to Web document analysis. In: Proceedings of the 1st international workshop on Web document analysis, Seattle, September 2001, pp 51–54
17. Lopresti D, Wilfong G (2001b) Comparing semi-structured documents via graph probing. In: Proceedings of the 7th International Workshop on Multimedia Information Systems, Capri, Italy, November 2001, pp 41–50
18. Lopresti D, Wilfong G (2001c) Evaluating document analysis results via graph probing. In: Proceedings of the 6th International Conference on Document Analysis and Recognition, Seattle, September 2001, pp 116–120
19. McKay B (1990) Nauty User's Guide (Version 1.5). Computer Science Department, Australian National University, Canberra, Australia
20. McKay B (1981) Practical graph isomorphism. Congressus Numerantium 30:45–87
21. Messmer BT, Bunke H (1995) Efficient error-tolerant subgraph isomorphism detection. In: Shape, Structure and Pattern Recognition, World Scientific, Singapore, pp 231–240
22. Myers R, Wilson RC, Hancock ER (2000) Bayesian graph edit distance. IEEE Trans Patt Anal Mach Intell 22(6):628–635
23. Nagy G, Seth S (1984) Hierarchical representation of optically scanned documents. In: Proceedings of the 7th International Conference on Pattern Recognition, Montréal, July 1984, pp 347–349
24. Ousterhout JK (1994) Tcl and the Tk toolkit. Addison-Wesley, Reading, MA
25. Papadopoulos AN, Manolopoulos Y (1998) Structure-based similarity search with graph histograms. In: Proceedings of the 10th International Workshop on Database and Expert Systems Applications, pp 174–178. IEEE Press, New York
26. Phillips I, Chen S, Haralick R (1993) CD-ROM document database standard. In: Proceedings of the 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japan, October 1993, pp 478–483
27. Sanfeliu A, Fu KS (1983) A distance measure between attributed relational graphs for pattern recognition. IEEE Trans Sys Man Cybern 13(3):353–362
28. Schlieder T, Naumann F (2000) Approximate tree embedding for querying XML data. In: Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval, Athens, Greece, July 2000, pp 53–67
29. Tsai WH, Fu KS (1979) Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans Sys Man Cybern 9(12):757–768
30. Valiente G, Martínez C (1997) An algorithm for graph pattern-matching. In: Proceedings of the 4th South American Workshop on String Processing, Valparaíso, Chile, November 1997, pp 180–197. Carleton University Press, Ottawa, Ontario

Daniel Lopresti received his bachelor's degree in mathematics from Dartmouth College in 1982 and his Ph.D. in computer science from Princeton University in 1987. He was on the faculty of the Computer Science Department at Brown University for 5 years before leaving to help found the Matsushita Information Technology Laboratory. After spending 5 years at MITL, he moved to Bell Labs, where he spent another 6 years as a researcher. He is now an associate professor in the Department of Computer Science and Engineering at Lehigh University in Bethlehem, PA. His research interests range broadly and have included work in VLSI systems, document analysis, pen computing, information retrieval, speech, and user interfaces and visualization techniques.

Gordon Wilfong received his B.Sc. in mathematics from Carleton University in 1980 and his master's and Ph.D. in computer science from Cornell University in 1982 and 1984, respectively. In 1984, he joined Bell Laboratories, where he is now a Distinguished Member of Technical Staff. He has been involved in research in the design of algorithms in areas such as scheduling, robotics, computational geometry, handwriting recognition, routing protocols, and optimal WDM cross-connect design.