Comparing Semi-Structured Documents via Graph Probing

Daniel Lopresti *Gordon Wilfong*

Bell Labs, Lucent Technologies Inc.
600 Mountain Avenue
Murray Hill, NJ 07974 USA
{dpl,gtw}@research.bell-labs.com

Abstract

In this paper, we describe our first steps towards adapting a new approach for graph comparison known as *graph probing* to allow for the pre-computation of a compact, efficient probe set for databases of graph-structured documents (*e.g.*, Web pages coded in HTML). We consider both the comparison of two graphs in their entirety, as well as determining whether one graph contains a subgraph that closely matches the other. After presenting an overview of work in progress, we provide some preliminary experimental results and suggest directions for future research.

1 Introduction

Graphs are a fundamental representation in much of computer science, including the analysis of both traditional and multimedia documents. Algorithms for higher-level document understanding tasks often use graphs to encode logical structure. HTML pages are usually regarded as tree-structured, while the WWW itself is an enormous, dynamic multigraph. Much work on attempting to extract information from Web pages makes explicit or implicit use of graph representations [3, 4, 7, 11, 13].

It follows, then, that the ability to compare two graphs is also basic functionality, as demonstrated by such diverse applications as pattern recognition, performance evaluation, query-by-structure, wrapper generation for information extraction, etc. Because most problems relating to graph comparison have no known efficient, guaranteed-optimal solution, researchers have developed a wide range of heuristics. For the problem of determining isomorphism, for example, many heuristics rely on the existence of certain *vertex invariants*, which consist of a value $f(v)$ assigned to each vertex v, so that under any isomorphism I, if $I(v) = v'$ then $f(v) = f(v')$. One commonly used vertex invariant is the degree of a vertex. It has been shown that for random graphs, there is a simple linear time test for checking if two graphs are isomorphic that is based on the degree of the vertices, and this test succeeds with high probability [1]. In fact **nauty**, a successful software package for determining graph isomorphism (see [10]), relies on such vertex invariants.

This observation can be seen as forming the basis for *graph probing*, a paradigm we have recently begun exploring for graph comparison [5, 6, 9]. However, we desire more than a simple "yes/no" answer; we are interested in quantifying the similarity between graphs, not just in whether they may be isomorphic. Conceptually, the idea is to place each of the two graphs under study inside a "black box" capable of evaluating a set of graph-oriented operations. We then pose a series of probes and correlate the responses of the two systems. This process is depicted in Figure 1.

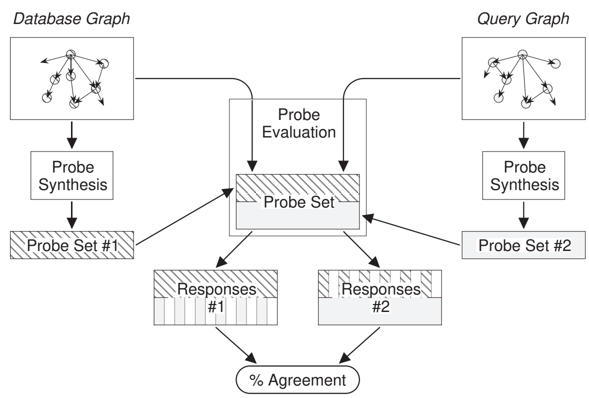

Figure 1: Overview of graph probing.

Our past work in the area treats graph probing as an on-line process; both the query graph and the database graph are available at run-time for synthesizing the probe set. While this is an appropriate assumption when one is comparing, say, the output of a recognition algorithm with its associated ground-truth, it is not a workable model for retrieval applications when the database contains anything other than a small number of documents.

In the present paper, we describe our first steps towards adapting the graph probing paradigm to allow for the pre-computation of a compact, efficient probe set for databases of graph-structured documents in general, and Web pages coded in HTML in particular. The comparison of two graphs in their entirety is considered, as well as determining whether one graph contains a subgraph that closely matches the other. We present an overview of work in progress, as well as some preliminary experimental results.

2 Related Work

Graph comparison is an important yet difficult problem, so it should come as no surprise that a large number of researchers have proposed heuristics or solutions designed for special cases. For example, Bunke and Messmer present a decision-tree-based precomputation scheme for solving the subgraph isomorphism problem [2], although their data structure can be exponential in the size of the database graphs in the worst case.

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 [8]. While they employ a similar set of features to the ones we use, they do not consider the approximate matching or subgraph problems.

Papadopoulos and Manolopoulos discuss an idea that is philosophically quite similar to ours [12]. 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 between HTML documents. We would like to determine a range of possible graph features that can be used to distinguish the sorts of effects that arise in practice. Moreover, their histogram technique is applied only to the problem of comparing complete graphs, whereas we wish to examine the subgraph matching problem as well.

Valiente and Martínez describe an approach for subgraph pattern-matching based on finding homomorphic images of every connected component in the query [14]. Again, the worst-case time complexity is exponential, but such features could also perhaps be incorporated in the heuristics we are about to present.

Instead of trying to solve the problem for graphs in general, some leeway can be had be 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 [13]. Likewise, Dubois *et al.* write about tree embedding for searching databases of semi-structured multimedia documents and for query-by-example [3].

The WebMARS system, as presented by Ortega-Binderberger *et al.* [11], models Web documents via their parse trees. Queries are likewise treated as trees, although they encode hierarchy for individual object types (*e.g.*, text, images) and do not represent the same sorts of inter-object relationships that the mark-up in Web documents encodes. Matching only takes place between the leaves of the query tree and all possible "chains" in the document tree (*i.e.*, paths leading in the direction from the root to a leaf). The match values are then propagated upwards towards the root of the query tree over edges that can be weighted to reflect the importance of that particular component. Hence, there is an asymmetry between queries and documents. In any case, the graph model we would like to support is more general than simple trees, allowing both cross-and back-edges.

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, which is a heuristic, 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, $dist^u(G_1^u, G_2^u)$. As noted previously, 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 "0"),

$$PR_{1a}(G^u) \equiv [\ |\{v \in V \mid deg(v) = n\}| \ \text{ for } n \in \{deg(v) \mid v \in V\}\ ] \tag{1}$$

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, within a factor of four, a lower bound on the true edit distance between any two graphs, $probe^u(G_1^u, G_2^u) \leq 4 \cdot dist^u(G_1^u, G_2^u)$.*

The time needed to perform the above probing procedure is $O(\max(|V_1|, |E_1|, |V_2|, |E_2|))$. In the off-line case, we can precompute the probes and their responses for one of the graphs. The result is exactly the same as in the on-line case. 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 particularly tight. Indeed, it is not even as tight as the bound that can be derived for the measure described in [12], although it does appear to be easier to generalize, a point that will become important shortly. Still, as a lower bound it does provide a potentially useful filter, which is our goal.

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?" Observations similar to those made above concerning lower bounds and computation time for undirected graphs apply here as well (including a theorem for directed graphs analogous to Theorem 1). It can also be shown that the bound returned by the new, more specific class of probes is at least as good as the original class and sometimes better: $probe^u(G_1^d, G_2^d) \leq probe^d(G_1^d, G_2^d)$.

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 HTML structure tags (*e.g.*, section heading, paragraph, table), while edges are labeled to represent relationships between structures (*e.g.*, contains, next, hypertext reference). 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 non-negative 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 $1 \leq i, j \leq \alpha$. 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. Define $probe^a(G_1^a, G_2^a) \equiv (PR_{1c}(G_1^a) - PR_{1c}(G_2^a)) + (PR_2(G_1^a) - PR_2(G_2^a))$. We then have:

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	$probe^a(G_1^a, G_2^a) \leq 4 \cdot dist^a(G_1^a, G_2^a)$

Table 1: Summary of the various models.

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

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

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 $|V|$ tuples can then be lexicographically sorted in $O(\alpha(d + |V|))$ time, where d 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 (non-empty) classes in additional time $O(\alpha|V|)$. Thus the total precomputation time is $O(\alpha(d + |V|) + |E|)$. Since α and d are likely to be small constants, the time is essentially the same as for the case of undirected graphs.

Table 1 summarizes the results of this section.

4 Graph Probing for Semi-Structured Documents

The attribute graph model we assume for HTML documents includes the standard tree-structured hierarchy generated when parsing the tags (the "contains"/"contained-by" relationship). In addition, we also make use of the order in which content and the various substructures are encountered (in many cases this corresponds to the natural reading order for the material in question). We represent this via "next"/"previous" cross-edges that connect vertices at a given level in the hierarchy, rather than assuming an implicit fixed ordering on the children of a vertex as some other researchers have done. Lastly, we record hyperlinks as either back-edges (in the case of targets on the same page) or a distinguished vertex type (in the case of external references). No provision is currently made for incoming links from outside documents. See Figure 4 for an example from our small test database.

Recall that we have defined two probe classes for these kinds of graphs:

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.

Class 2 These probes count the occurrences of a given type of vertex in the graph.

When comparing two graphs in their entirety, it suffices to correlate their responses to the probes using the L_1 norm as described earlier. For the problem of subgraph matching, however, we cannot expect to be able to compare directly the outputs for the larger graph to those for the smaller. For example, consider a query graph consisting of a single table that corresponds to a table in some database document, but where that document also has dozens of other, unrelated tables.

In the case of the Class 2 probes, clearly if the query graph contains a certain number of vertices labeled in a given way, the target graph may possibly contain a perfectly matching subgraph so long as it contains at least that many vertices labeled in the same way. Similarly, the way in which the Class 1c probes are correlated needs to be modified as well, since the vertices present in the query graph may have fewer incoming and outgoing edges than their corresponding vertices in a matching subgraph of a larger graph.

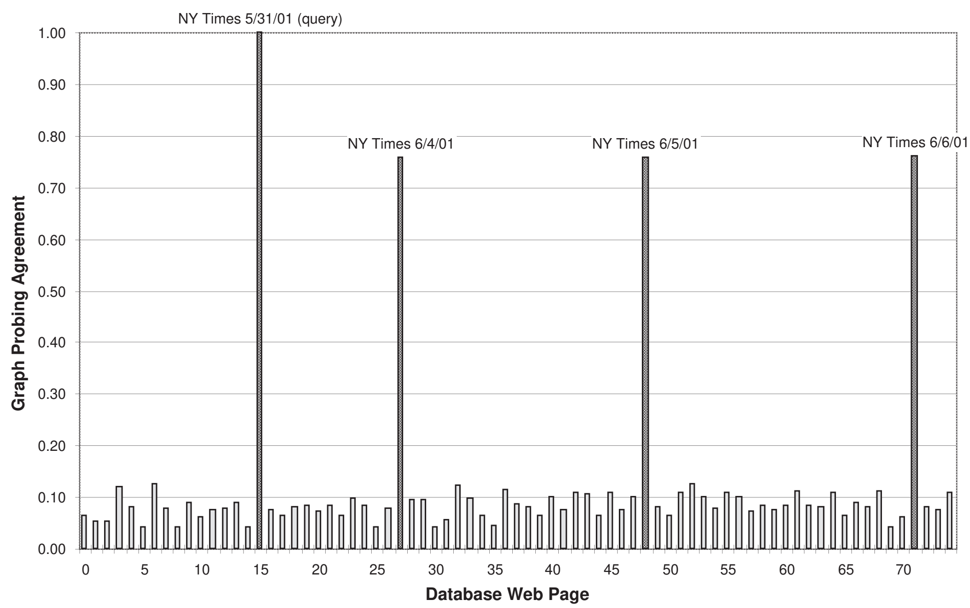

Figure 2: Graph probing results for Experiment 1 (full graph comparison measure).

5 Preliminary Experimental Results

This paper describes research that is still very much in progress. As we noted, we have previously implemented the on-line version of graph probing (Figure 1) to help in the evaluation of table understanding algorithms [5, 6]. However, the utility of graph probing in that specialized domain is not an assurance that it is an appropriate paradigm for searching databases for matches in a retrieval application, especially since the classes of probes we have at our disposal are also different here.

To begin examining some of these issues, we performed two simple experiments to test graph probing in an information retrieval setting. In both cases we used Class 1c and Class 2 probe sets. The first test examined the ability of the two classes to detect changes as the structure of a specific commercial Web page evolved naturally over several days. The second studied the subgraph matching problem by searching for a Web page that had been edited by deleting a significant portion of its content.

The small database consisted of 75 current WWW homepages collected from a variety of sources: commercial, educational institutions, personal, news organizations, and portal pages. Our parse graph generator is capable of recovering from the kinds of simple errors that often arise in real-world HTML (*e.g.*, missing end tags). The size of the resulting graphs ranged from a minimum of 60 vertices to a maximum of 1,204, with an average of 419 vertices. The probe set, generated automatically, consisted of a total of 219 Class 1c and Class 2 probes.

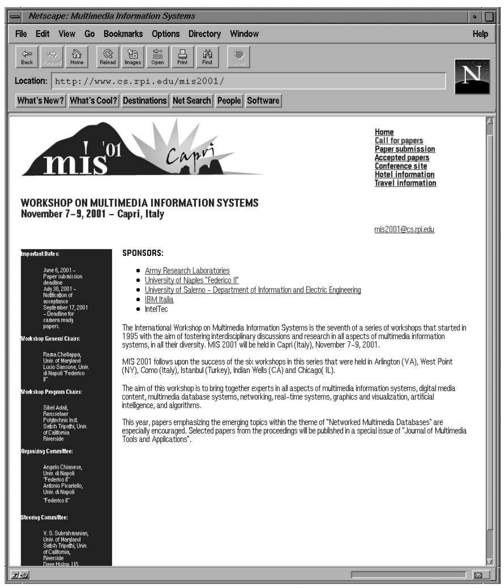

Figure 3: Screen snapshot of the MIS'2001 homepage (http://www.cs.rpi.edu/mis2001/).

For the first experiment, we used as our query the May 31, 2001 homepage from *The New York Times* (http://www.nytimes.com/). This is a relatively complex page, its parse graph containing 1,089 vertices (for comparison, the graph depicted in Figure 4 is less than one-fifth this size). The results for using graph probing to compare this page to the 75 pages in the database are shown in Figure 2. The May 31 page is, of course a perfect match for itself, but also a very good match for the June 4, 5, and 6 homepages as well (the only other examples from *The New York Times* in the database). Clearly some sort of structural change in the page was made between May 31 and June 4. Conversely, a number of other regularly-updated Web pages we have examined show no structural changes from day-to-day, although obviously the content is constantly varying.

For our second experiment, we used the homepage for the MIS'2001 workshop, a screen snapshot of which is given in Figure 3. The corresponding parse graph, containing 193 vertices, is shown in Figure 4. To create the query, the page was edited by deleting the dark blue sidebar on the left side of the page. The parse graph for the edited page had 124 vertices and appears in Figure 5.

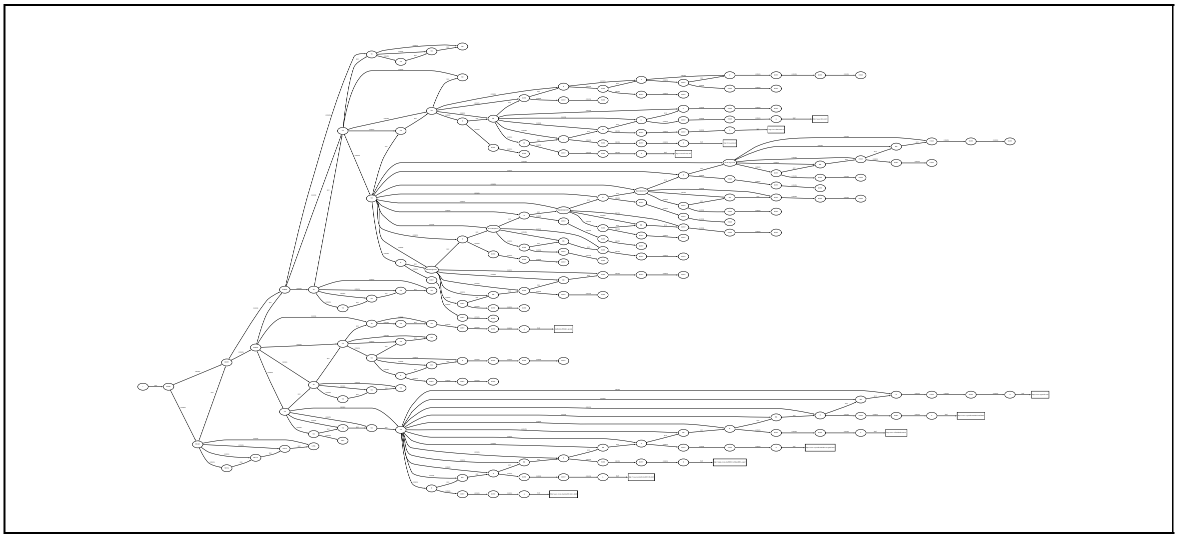

Figure 4: Parse graph for the MIS'2001 homepage (193 vertices).

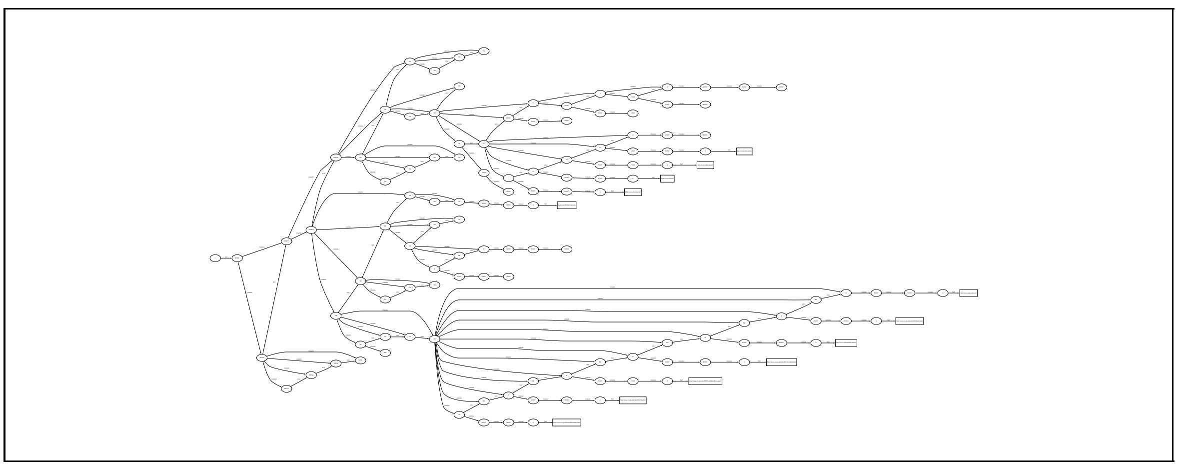

Figure 5: Parse graph for edited version of the MIS'2001 homepage (124 vertices).

The results for this experiment, using graph probing with the subgraph comparison measure, are shown in Figure 6. Here we can see that the two probe classes are able to distinguish the original page and the smaller, edited version from the rest of the database, but just barely. The current probes, which consider only structure (and high-level structure at that), need to be supplemented with new classes that are able to make use of content and other aspects of the page layout before the probing paradigm can be effective for the subgraph matching problem. This is work in progress.

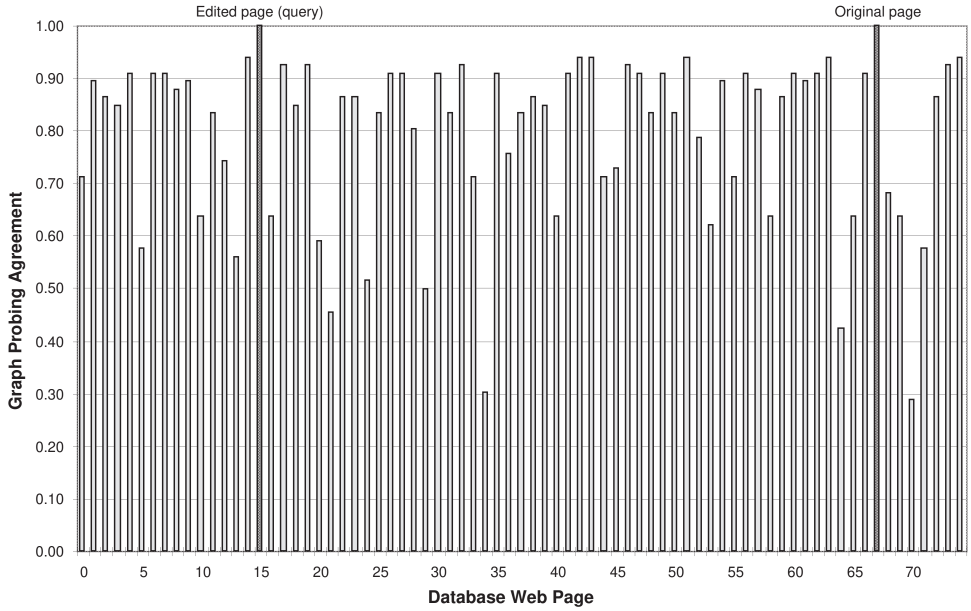

Figure 6: Graph probing results for Experiment 2 (subgraph comparison measure).

6 Discussion

In this paper we have described our initial efforts to adapt the graph probing paradigm to searching databases of graph-structured documents. We considered both the comparison of two graphs in their entirety, as well as the subgraph matching problem, and gave some preliminary experimental results.

Currently, our graph probing provides a measure of how similar two graphs are or how similar one graph is to some subgraph of another. It does not, however, produce a mapping from one graph to the other. Clearly, to extract information from semi-structured sources we need to recognize more than the fact that some matching is likely to exist; we must be able to identify the actual correspondence between the graphs (or between one graph and a subgraph of the other). At the very least, however, graph probing can be used as a filter to exclude graphs that could not possibly be a good match so that computationally expensive methods may be run on much smaller collections of potential candidates.

Another topic for future research is extending the formalism and bounds of Section 3 to the subgraph matching case.

References

[1] L. Babai, P. Erdös, and S. M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, August 1980.

[2] H. Bunke and B. T. Messmer. Recent advances in graph matching. *International Journal of Pattern Recognition and Artificial Intelligence*, 11(1):169–203, November

1997.

[3] D. Dubois, H. Prade, and F. Sèdes. Some uses of fuzzy logic in multimedia databases querying. In *Proceedings of the Workshop on Logical and Uncertainty Models for Information Systems*, pages 46–54, London, England, July 1999.

[4] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(3):59–74, 1998.

[5] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. A system for understanding and reformulating tables. In *Proceedings of the Fourth IAPR International Workshop on Document Analysis Systems*, pages 361–372, Rio de Janeiro, Brazil, December 2000.

[6] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table structure recognition and its evaluation. In *Proceedings of Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging)*, volume 4307, pages 44–55, San Jose, CA, January 2001.

[7] N. Kushmerick. Regression testing for wrapper maintenance. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 74–79, Orlando, FL, 1999.

[8] M. Lazarescu, H. Bunke, and S. Venkatesh. Graph matching: Fast candidate elimination using machine learning techniques. In *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 236–245. Springer-Verlag, Berlin, Germany, 2000.

[9] D. Lopresti and G. Wilfong. Evaluating document analysis results via graph probing. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 116–120, Seattle, WA, September 2001.

[10] B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[11] M. Ortega-Binderberger, S. Mehrotra, K. Chakrabarti, and K. Porkaew. WebMARS: A multimedia search engine for full document retrieval and cross media browsing. In *Proceedings of the Sixth International Workshop on Advances in Multimedia Information Systems*, pages 72–81, Chicago, IL, October 2000.

[12] A. N. Papadopoulos and Y. Manolopoulos. Structure-based similarity search with graph histograms. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pages 174–178. IEEE Computer Society Press, 1998.

[13] T. Schlieder and F. Naumann. Approximate tree embedding for querying XML data. In *Proceedings of the ACM SIGIR Workshop On XML and Information Retrieval*, Athens, Greece, July 2000.

[14] G. Valiente and C. Martínez. An algorithm for graph pattern-matching. In *Proceedings of the Fourth South American Workshop on String Processing*, pages 180–197. Carleton University Press, 1997.