Efficient Content-Based Retrieval: Experimental Results

Andrew P. Berman and Linda G. Shapiro
Department of Computer Science and Engineering
University of Washington
PO Box 352350
Seattle, WA 98195
aberman@cs.washington.edu and shapiro@cs.washington.edu

1 Introduction

The goal of our research has been to create technology useful in a generalized system for content-based image retrieval. Such a system should search and retrieve images quickly, and do so over a wide variety of queries. As user definitions of similarity may change from session to session, we believe that flexibility in query formulation is an important quality for content-based retrieval systems. This flexibility can be achieved by providing the user with a large set of distance measures that determine the similarity between a user query and a database image and a small set of methods for combining several of them for a trial query. Distance measure calculation requires accessing either the images being compared, or pre-computed associated data. An image database can consist of millions of images, each one taking many megabytes of storage space. Distance measure calculations can be individually expensive as well. Thus, a system that must calculate the distance from the query image to each image in a large database may exhibit unsatisfactory performance.

For certain distance measures and data sets, indexing or clustering schemes can be used to reduce the number of direct comparisons. For example, one could order images based on overall brightness to conduct fast searches based on similar brightness. But such schemes are highly dependant on internal characteristics of the particular distance-measure algorithms, and do not serve the needs of a more general-purpose system.

We have designed and implemented a prototype database system that allows the user a great deal of flexibility in run-time distance measure creation. The system can also reduce the number of direct distance measure calculations between a given query object and the database elements. FIDS, or “Flexible Image Database System,” has been tested on a database of thirty seven thousand images. FIDS allows the user to find approximate matches to query images using complex combinations of dozens of pre-defined distance measures. FIDS can often return results without directly comparing the query image to more than a small percentage of the original database. In Figure 1, FIDS was presented with the query image at the far left. The figure shows the results of FIDS searching through $37,748$ images to find close matches to the query, using a texture-based distance measure. The operation took just a few seconds, and only used 55 direct comparisons to find the results.

There are algorithms in the literature based on the triangle inequality that can reduce the number of distance measure calculations[1, 2, 3, 6, 8] in object retrieval. These methods have the advantage of being applicable to any distance measure that satisfies the triangle inequality. FIDS uses algorithms based on the triangle inequality that are extensions to the methods in the literature. We have published papers that described the use of the triangle inequality for efficient multiple-distance-measure retrieval[6], the selection of key images for indexing[4], the additional use of a data structure called the *triangle trie* to make retrieval time sublinear[7], and the details of the FIDS system[5]. In this paper, we briefly summarize those aspects and present a set of experiments that thoroughly evaluates the FIDS techniques and system.

2 The use of combinations of multiple distance measures

A distance measure is a function that computes and returns a value corresponding to the similarity between two objects according to the predefined criteria. Formally, function $d(X,Y)$ with non-negative real values is a *metric* if it satisfies the identity axiom: $d(X,Y) = 0 \iff X = Y$, the symmetry axiom: $d(X,Y) = d(Y,X)$, and the triangle axiom: $d(X,Y) + d(Y,Z) \geq d(X,Z)$. Relaxing the

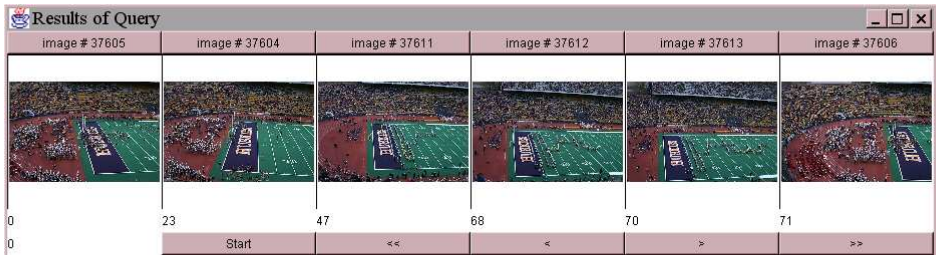

Figure 1. Result of a query submitted to FIDS. The query image is on the far left.

identity axiom so that $d(X, Y)$ may be equal to 0 when $X \neq Y$ yields the definition of a *pseudo-metric*. The distance measures discussed in this paper are all assummed to be pseudo-metrics.

One can create an immense number of distance measures for images based on any set of features and an arbitrary scoring mechanism. We cannot program all these distance measures in advance. This difficulty motivates the idea of giving the user a pre-defined set of base distance measures that he or she can combine to create more complex measures. We proposed[5] the following set of operations to enable more expressive queries ($d_1 ... d_n$ represent distance measures): addition: $d = d_1 + d_2$; weighting: $d = cd_1, c \geq 0$; max: $d =$Max$(d_1, d_2, \dots, d_n)$; and min: $d =$Min$(d_1, d_2, \dots, d_n)$. These operations are all invariant under inequality. Weighting and addition are already commonly used in both commercial and research systems [11, 10]. Min and Max allow expressions with the boolean operators AND and OR. Fagin[9] has also proposed extending multimedia queries to boolean combinations, using Min and Max to implement them.

3 Indexing with the triangle inequality

Traditional database searches are based upon retrieval of entries which contain a keyword that is a precise match to the user's query. There have been a host of algorithms developed for data structures such as hash tables and B-trees that can be used in exact match searches. In contrast, a content-based retrieval system using similarity measures assumes no exact matches. Thus, the well-known algorithms for exact matching cannot be used. There is a need to develop robust data structures and algorithms that can handle searches for inexact matches efficiently.

The indexing scheme and algorithm described here, defined informally as the *bare-bones triangle inequality algorithm*, outputs a value for each database image corresponding to a lower bound on the distance between that image and the query image. This set of values can be used in several different ways: to discard images that are shown to be too far from the query image to be a potential match or to sequence the images in increasing order of their calculated lower bounds.

Let I represent a database object, Q represent a query object, K represent an arbitrary fixed object known as a *key*, and d represent some distance measure that is a metric. As d is a pseudo-metric, the two triangle inequalities, $d(I, Q) + d(Q, K) \geq d(I, K)$ and $d(I, Q) + d(I, K) \geq d(Q, K)$, must be true. We can combine them to form the following inequality, which places a lower bound on $d(I, Q)$:

$$d(I, Q) \geq |d(I, K) - d(Q, K)| \tag{1}$$

Equation (1) can be extended naturally by substituting a set of keys $\mathcal{K} = (K_1, \dots, K_M)$ for k as follows:

$$d(I, Q) \geq max_{1 \leq s \leq M} |d(I, K_s) - d(Q, K_s)| \tag{2}$$

Consider a large set of database objects, $\mathcal{S} = \{I_1, \dots, I_n\}$ and a much smaller set of key objects, $\mathcal{K} = \{K_1, \dots, K_m\}$. Pre-calculate $d(I_s, K_t)$ for all $\{1 \leq s \leq m\} \times \{1 \leq t \leq n\}$. Now consider a request to find all database objects I_s such that $d(I_s, Q) \leq t$ for some query image Q and threshold value t. We can calculate lower bounds on $\{d(I_1, Q), \dots, d(I_n, Q)\}$ by calculating $\{d(Q, K_1), \dots, d(Q, K_m)\}$ and repeatedly using equation (2). If we prove that t is less than $d(I_s, Q)$, then we eliminate I_s from our list of possible matches to Q. After the elimination phase, we may search linearly through the uneliminated objects, comparing each to Q in the standard fashion.

We extended the above scheme to work with combinations of distance measures[6]. The intuition is that lower bounds on the distance between two objects for distance measures d_1 and d_2 can often be used to calculate a lower bound between the objects for distance measure d when d can be calculated as a combination of d_1 and d_2. Our results show that the type of elimination described above can be extended to any monotonically non-decreasing func-

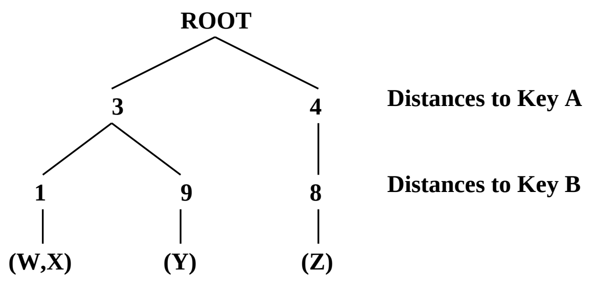

Figure 2. Triangle Trie with two levels.

tion f whose inputs are the outputs of single distance measures. We have applied this to expressions involving Addition, Weighting, Max, and Min.

4 The triangle trie

Although much faster than direct comparisons, the basic triangle-inequality algorithm described above has a running time of $O(nk)$, where n is the number of images and k is the number of keys. Running time may become unacceptable for very large databases with a large number of keys. Therefore, we take advantage of a data structure called the *triangle trie*[3] to reduce the number of operations.

A *triangle trie*, otherwise known as a *Really Fixed Query Tree*[1], is a data structure developed for approximate-match searching. A single triangle trie is associated with a distance measure, a set of key images, and a set of database elements. It is a form of *trie*, a tree in which the edges leading from the root to a leaf "spell out" the index of the leaf. Figure 2 illustrates a triangle trie with four elements (W, X, Y, Z), and two keys (A, B). The distance from W to A is 3 and the distance from W to B is 1. This is expressed in the trie by the path from the root to the leaf containing W.

Suppose we are given query Q and threshold integer t and wish to find all objects in our database with a distance from Q of not more than t. Now, consider a node P at level l with a value of c. Every object referenced in a leaf descendant from P has a distance of c from the key object K_l. Thus, if $|c - d(Q, K_l)|$ is greater than t, then we know from the triangle inequality that $d(Q, I)$ is greater than t for all object I which are descendants of P. Thus, we can safely prune the search at node P.

The algorithm for searching the database using the triangle trie is straightforward. Compute the distances from Q to each key: $d(Q, K_1), \ldots, d(Q, K_m)$. Perform a depth-first search of the trie. If there is a node P at level l with value c such that $|c - d(Q, K_l)| > t$, then prune the search at node P. When a leaf is reached, measure the distance from Q to every object in the leaf and return those objects I for which $d(Q, I)$ is less than or equal to t. There may be cases where the distance measure is not integer valued, or where the distance measure has such a wide variance that any resultant triangle trie would have a very quick fan-out. In these cases, it may be necessary to map the calculated distances to a smaller set of values. We call this process *binning* the distances. As binning a distance may reduce its accuracy, there may be occasional lost opportunities for pruning.

The breadth of a triangle trie expands with its depth, up to a maximum breadth equal to the number of database elements. Our work suggests that by using a relatively short trie, and by storing additional key distances in the leaves, we can achieve excellent performance with bounded memory use. Our two-stage algorithm works as follows: Given database images $\{I_1, \ldots, I_n\}$, keys $\{K_1, \ldots, K_m\}$, and distance measure d, we create a triangle trie T of depth T_{depth} where $T_{depth} < m$. Given query Q, we perform our search of the trie as described above. This trie traversal is the first stage. The second stage is to run the the bare-bones triangle inequality algorithm on the remaining images using all the keys. Figure 3 shows an example in which a color measure and threshold were used to eliminate all but 19 of the $37,748$ images from contention as potential matches to the query image. However, $37,748$ lower bounds need to be calculated to eliminate these images without the use of a triangle trie. By using a triangle trie with 6 levels, we reduced the number of calculated lower bounds to $7,448$, an 80 percent reduction.

A triangle trie is designed to enable thresholded database searches for a single distance measure. However, it is possible to use multiple triangle tries to enable thresholded database searches over a composite measure. Searches are done on the individual tries and the returned sets of images are either merged or intersected, depending on how the distance measures are composited.

5 Fast Image Database System: A prototype for testing

FIDS, the Fast Image Database System, is a prototype content based image retrieval system. It currently has over thirty seven thousand images. It contains a number of distance measures based on color, texture, and feature detection. Position matching is implemented by providing gridded versions of basic distance measures. The default configuration for the system returns the images in order of their lower bounds calculated by the two-stage pruning algorithm. Optionally, the system can be configured to calculate the true distance from the query to the initial images in the returned list, providing for a more accurate

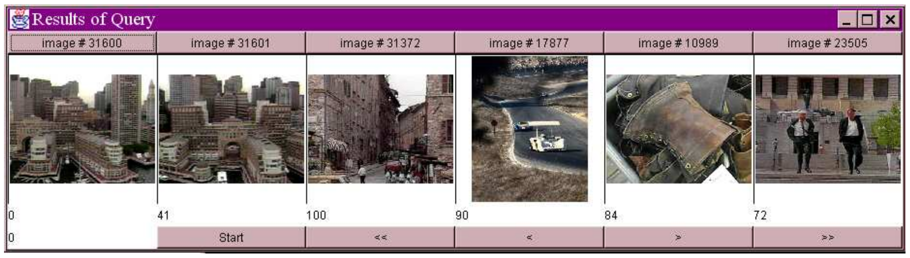

Figure 3. An example using a simple color measure. The query image is on the far left.

result. FIDS is currently located on a Pentium-II 200 mhz machine running the Windows NT operating system. The machine contains 192 megabytes of RAM. FIDS is written in Java version 1.1, and consists of approximately 7000 lines of source code. The images in the database consist of 37748 images taken from a commercial CD-ROM image gallery published by $IMSI$ plus an additional 150 images taken as part of the *Ground-Truth Image Database* project initiated at the University of Washington. The sizes of the uncompressed images range from 320 by 240 pixels to 1200 by 960 pixels. They are stored in JPEG format on a local hard drive, taking about 900 megabytes of space.

FIDS has a set of *base* distance measures which can optionally be combined into composite measures. Each base measure is linked with a set of keys K, a triangle-trie T, and a table containing the distances from each of the database images to each of the keys in K. Keeping the images associated with K in memory would be wasteful. Rather, the features necessary for distance comparison are stored.

6 Experiments and results

We conducted a number of experiments on various aspects of the algorithms and system.The experiments were performed using various subsets of the following distance measures: 1) a color-histogram measure, 2) the local-binary-partition texture measure, 3) a Sobel-edge-histogram texture measure, 4) a wavelet representation comparison measure, 5) a flesh detection measure, 6) gridded versions of these measures, and 7) combinations of 1-6 using SUM, MIN, and MAX to form the composite functions.

The speed of the bare-bones triangle inequality algorithm

We first measured the speed of the bare-bones triangle inequality algorithm. The process by which a system using this algorithm returns a set of matches to a query can be broken into four steps:

- **Step 1:** The system extracts relevant features from the query image. In our system, this step takes from a fiftieth to a quarter of a second, depending on the distance measure. Table 1 shows this range in the second column.

- **Step 2:** the system calculates the distance from the query image to each of the key images. For the basic distance measures on our system, this step takes from about a microsecond per image to more than four-fifths of a millisecond per image. The third column of table 1 shows the values.

- **Step 3:** The system calculates the lower bound distances from the query image to each of the database images.

- **Step 4:** The system returns the images with the smallest lower bound distances calculated in the previous step.

The third step above is the deciding factor on throughput. The timing of the other steps is relatively stable across database sizes, although the number of keys in step two which are necessary for adequate performance will tend to increase as the number of images in the database increases. We measured a time of approximately 4 milliseconds to perform the third step on 1,000 images if 35 keys are used. This represents a throughput of well over two hundred and fifty thousand images per second, once the feature extraction and query-key comparisons are finished. By comparison, the final column of Table 1 gives the throughput per second for each distance measure given a system that stores the features for its images.

Two stage pruning with triangle tries

We next tested the performance of the triangle trie in image searches. We created a set of triangle tries for each distance measure, varying the tries by depth and bin size. The depths ranged from 1 to 11, and the bin size ranged from 10 to 130 stepping by 10.

Distance Measure	A	B	C
Color (4x4x4 RGB)	.12	46	2174
Color (8x8x8 RGB)	.13	37	2669
LBP	.04	28	3623
Flesh	.25	.12	833333
Sobel	.20	4.1	24937
Wavelet	.02	863	115

Table 1. Feature extraction and distance calculation time for representative distance measures used in FIDS on a Pentium Pro 200 mhz PC.
A: Feature Extraction Time in Seconds
B: Distance Calculation Time in ms/1000
C: Number Of Distance Calculations per Second

There are 35 key images in our database. Any subset of them can be used as the keys for the tries. We created five random orderings of the keys. For each triple of 10 distance measures, 11 depths, and 13 bin sizes, we created five tries using the appropriate five prefixes of the random key orderings. For each such trie, we tested five images, resulting in a total of twenty-five tests for each triple of distance measure, depth, and bin size. This resulted in a total of $33,295$ experiments, not including experiments that were abandoned due to too large a trie being created. Each experiment was repeated 250 times to improve the accuracy of the measured time. We recorded the size of the created tries, the number of matches returned for a search, the number of internal trie nodes accessed, the number of leaf trie nodes accessed, and the time it took to walk the trie and return the matches.

Table 2 shows the results of the tries that had the highest speed improvement factors. The gridded version of the local-binary-partition texture measure had the minimum improvement, being about twice as fast with a trie as without one. At the other end of the scale, the Sobel-edge-histogram texture measure was almost fifty times as fast with a trie as without one. This represents a potential processing rate of almost twelve million images per second. The Triangle Trie offered improved performance on all of the tested distance measures.

The accuracy of the triangle-inequality algorithms

To evaluate the efficiency of using the triangle inequality to return close matches to queries, we performed experiments using both single and composite distance measures. Given a query and distance measure, the system returns all the images in the database ordered by calculated lowest bounds on their distances to the query. We use terms such as *lower-bound sequence* or *returned ordering* to refer to

Measure	A	B	C	D	E
Sobel	47.8	10	6	0.4	318.6
Gridded Sobel	38.6	30	15	0.7	343.0
Gridded Wavelets	22.3	20	8	1.2	598.6
Gridded Color	15.01	130	11	1.8	882.4
Color	8.8	80	8	4.0	1273.0
Wavelets	3.8	40	4	11.8	2343.6
Gridded Flesh	3.2	30	4	7.0	4588.2
LBPHist	2.8	30	7	20.1	5740.7
Flesh	2.6	80	11	7.4	5980.2
Gridded LBPHist	2.4	80	8	12.1	5465.0

Table 2. Characteristics of the best tries found for each distance measure, rated by speed factor. Time is in milliseconds for a search of 20,000 images and is just for the trie search itself. Speed Factor is the ratio of time taken for a search of 20,000 images using the triangle inequality algorithm without the triangle trie to a search using the triangle trie as a precursor to the triangle inequality algorithm.
A: Speed Factor
B: Bin Size
C: Trie Depth
D: Time
E: Returned Matches

the system's output. On the other hand, there is the *true sequence* or *true ordering*, which consists of all the images in the database ordered by their true distance to the query. The hope is that the images which are at the front of the true ordering are also at or near the front of the lower-bound sequence. Measuring the placement of close matches was the main goal of these experiments.

In order to test the quality of the returned ordering, some ground truth was required. To achieve this ground truth, we examined the database by hand and selected 51 pairs of similar images. For each pair, we queried the system with one of the images and measured the position of the other *target* image in the returned sequence. We then calculated the true position of the target image and compared the returned positions to the true positions.

We measured the fraction of matches that were returned as part of the first 25 images. This corresponds to a scenario where a user might ask for several images to be returned for closer examination. We further measured the fraction of matches that were returned within the first 400 matches, corresponding to a scenario in which the user or system is willing to do more work to find a closer match. We also measured the fraction of images that were returned in their precise position. As we selected matching pairs of test images by hand, we had to deal with the possibility that the test image pairs didn't truly match with respect to the distance measures on the system. Our experiments were designed to measure the ability of the system to find the closest images to queries, so we excluded an image pair from tests on a dis-

Distance Measures	Percent in first 400		
	100%	90-99%	80-89%
Single Measures	3	2	3
AND'ed Measures	7	12	1
OR'ed Measures	16	1	3
SUM'ed Measures	10	10	0
Totals	36	25	7

Table 3. Summary performance of distance measures using full set of validated images. There were no distance measures in which less than 80% of the target images were returned in the first 400 images out of 37,748.

Distance Measures	Percent in first 25				
	100%	90-99%	80-89%	70-79%	50-69%
Single Measures	1	0	1	4	2
AND'ed Measures	2	2	10	6	0
OR'ed Measures	6	0	0	12	2
SUM'ed Measures	4	6	10	0	0
Totals	13	8	21	22	4

Table 4. Summary performance of distance measures using full set of validated images, showing how many distance measures of each type returned the target images in the first 25 images.

tance measure if it was discovered that the target image was not one of the five actual closest images to the query image for that distance measure. Our database contained $37,748$ images and we used 35 keys for each base measure.

Results

Tables 3 and 4 show the results of experiments using the full set of valid matches. Depending on the distance measure, from 50% to 100% of all the matches were within the first 25 images returned by the system. In almost every case, over 90% of the matches were within the first 400 returned images. In a second set of similar experiments, we simulated a user with a more restricted definition of closeness. For each distance measure, the query-target pairs were sorted in increasing order of query-target distance. Only the first half of these lists were used to compute these results. The results were markedly better for the experiments using the closer half of the valid matches. Tables 5 and 6 show the results of experiments with the restricted set of valid matches. In this case, only 9 distance measures out of the 68 tested measures

Distance Measures	Percent in first 400			
	100%	90-99%	80-89%	70-79%
Single Measures	6	0	2	0
AND'ed Measures	20	0	0	0
OR'ed Measures	18	1	0	1
SUM'ed Measures	20	0	0	0
Totals	64	1	2	1

Table 5. Summary performance of distance measures using closer pairs of validated images. Almost every distance measure returned all targets within the first 400 images.

Distance Measures	Percent in first 25				
	100%	90-99%	80-89%	70-79%	50-69%
Single Measures	2	1	2	1	2
AND'ed Measures	13	5	0	2	0
OR'ed Measures	6	0	10	2	2
SUM'ed Measures	16	3	1	0	0
Totals	37	9	13	5	4

Table 6. Summary performance of distance measures using closer pairs of validated images, showing how many distance measures of each type returned the target images in the first 25 images.

had less than 80% of the target images returned within the first 25 images. Furthermore, in 64 out of 68 distance measures, every single one of the matches were returned within the first 400 images.

7 Summary

Extensive testing has shown that the bare-bones triangle inequality algorithm could be used to sharply reduce the number of images needed to be directly compared to a query image for a given distance measure, and that adding the triangle-trie for a two-stage algorithm can be used to search for matches faster than even the bare-bones triangle inequality algorithm. We have developed a method for using the triangle inequality algorithm for combinations of distance measures, thus allowing for database systems which combine flexibility and speed.

There are a number of open problems concerning the various data structures and algorithms we described. We have already mentioned some of them, like key selection, number of keys, trie depth and bin size. More generally, the statistical behavior of distance measures over different sets of images influences the behavior of all the algorithms and thus needs to be explored.

References

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Combinatorial Pattern Matching*, pages 198–212. Springer-Verlag, June 1994.

[2] J. Barros, J. French, W. Martin, P. Kelley, and M. Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval. In *IS&T/SPIE - Storage and Retrieval for Still Image and Video Databases*, volume IV, Jan 1996.

[3] A. Berman. A new data structure for fast approximate matching. Technical Report 1994-03-02, Dept. of Computer Science, University of Washington, 1994.

[4] A. Berman and L. Shapiro. Selecting good keys for triangle-inequality based pruning algorithms. In *IEEE International Workshop on Content-based Access of Image and Video Databases*, 1997.

[5] A. Berman and L. G. Shapiro. A flexible image database system for content-based retrieval. In *17th International Conference on Pattern Recognition*, 1998.

[6] A. P. Berman and L. G. Shapiro. Efficient image retrieval with multiple distance measures. In *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, February 1997.

[7] A. P. Berman and L. G. Shapiro. Triangle-inequality-based algorithms with triangle tries. In *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, January 1999.

[8] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, Apr 1973.

[9] R. Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–10, June 1998.

[10] M. Flickner, H. Sawhnew, W. Niblack, J. Ashley, Qian-Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the qbic system. *Computer*, 28(9):23–32, Sep 1995.

[11] A. Gupta. Visual information retrieval: A virage perspective. Technical report, Virage, Inc. http://www.virage.com/wpaper, 1995-1997.