Heuristic Clustering of Signature Files: A New Approach (Preliminary Results)

James C. French

Technical Report No. CS-92-42
September 29, 1992

This work was supported in part by JPL Contract #957721, Dept. of Energy grant DE-FG05-88ER25063, Simpson Weather Associates, and the Virginia Center for Innovative Technology under grant VCIT CAE-91-012.

Heuristic Clustering of Signature Files: A New Approach (Preliminary Results)

James C. French
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

Abstract. Online full-text databases are placing increasing demands on computing resources. Efficient mechanisms are necessary to manage the storage and retrieval of text in these very dynamic environments. This paper discusses a new approach to clustering signature files which provides excellent retrieval performance at a low computational cost during insertion.

1. Introduction.

Full-text databases are becoming increasingly available in all facets of information storage and retrieval. Wire services maintain enormous archives of news articles. Legal statutes and opinions, scientific and technical journals, books — all are currently available online. This is no longer the exclusive purview of commercial concerns. Individuals are maintaining and contributing vast quantities of full-text to distributed or wide-area information services [LEVI91]. The office environment provides abundant opportunity to collect text and file it electronically. The continued growth of online full-text databases is placing increasing demands on resources necessary to store and retrieve the documents.

As online databases have grown, so too have the technologies for information retrieval. Many alternative file structures and organizations have been proposed to manage and access these text databases [FALO85]. One technique that has been proposed, signature files, provides for the search and retrieval of texts by examining hash coded representations of the documents. This paper examines a new method for organizing signature files to achieve better retrieval performance.

This work was supported in part by JPL Contract #957721, Dept. of Energy grant DE-FG05-88ER25063, Simpson Weather Associates, and the Virginia Center for Innovative Technology under grant VCIT CAE-91-012.

2. Signature Files.

Signature files in many variants have been proposed for full-text retrieval applications. The basic idea is that each document to be stored has a signature created for it. The signatures for all the documents in the system are used to speed subsequent retrieval by rapidly reducing the document search space.

Document signatures can be formed in a variety of ways. The precise technique used is not important to the results of this paper. It suffices to consider the signature as a fixed length bit string. To make this more concrete we show an example of signature formation using superimposed coding [ROBE79] in Fig. 1.

In this example we are assuming that the document consists of the three words *quick*, *brown* and *fox* and the signature is 8 bits in length. Conceptually, each word is hashed to set 3 of the 8 bit positions. After these word signatures have been formed, they are superimposed by inclusive-OR to form the document signature. Now consider a query on one of the words in the document. The query word is hashed and each of the 3 bits set in the query signature is also set in the document signature so the document will be retrieved. If the word *hen* is used, then Query 1 results and because only two of the bits match the document signature, the document is not fetched. It is this filtering behavior that makes signature files attractive. A relatively inexpensive test is used to screen documents. Unfortunately, this screen is not perfect and may be confounded as shown by Query 2. A search using the word *egg* causes the document

Example of Signature Formation

Document	quick	01001100
	brown	11000100
	fox	10001001
Signature		11001101
$Query_1$	hen	01101000
$Query_2$	egg	11000001

Figure 1

to be fetched even though the word does not appear in the document. This phenomenon, known as a false drop, occurs because of the superimposition of the word signatures. False drops may adversely affect retrieval performance so they must be kept acceptably low.

All the document signatures are kept together in a signature file. Query signatures are compared with the signature file and a document is fetched whenever each bit in the query signature is also set in the document's signature. The signature file is examined in its entirety and qualifying documents are examined to determine if the query terms actually occur in the documents.

Many variants on the organization and processing of signature files have been discussed in the literature. The simplest organization is a sequential file (also called the bit-string organization). If we consider the signature file to be a binary matrix then it is clear that this organization is optimized for row-wise access. The bit-slice approach [ROBE79] seeks to improve performance by accessing the columns of the matrix. In this case the transpose of the sequential file is stored. The bit-slice approach allows the access of the particular bit positions specified by the query and only those positions.

Various multilevel and tree-structured organizations have been proposed. Pfaltz et al. [PFAL80] and French [FREN85] discuss multilevel tree structures. Sacks-Davis [SACK83] proposed a two-level structure (one level stored as bit-strings and one level stored as bit-slices) and its variant the multiorganization scheme [KENT88]. A dynamic tree with properties analogous to B-trees has been described by Deppisch [DEPP86]. Kotamarti and Tharp [KOTA90] have also discussed text searching in signature trees. These structures all have as a common goal the reduction of the signature search space.

Signature file organizations based on clustering the signatures have also been proposed. These are discussed in the next section.

3. Clustering Signature Files.

Document clustering [CAN90, SALT78, WILL88] seeks to improve the *effectiveness* of information retrieval systems by grouping documents so that those likely to be relevant to particular queries are in the same groups. Signature clustering seeks to improve retrieval *efficiency* by grouping signatures so that

those matching any particular query fall into ideally one, but at most, a small number of groups. When a query signature is processed sequentially against a signature file of N signatures, it will match $m \geq 0$ signatures. Intuitively, if these m signatures were present in a single cluster and that cluster could be identified quickly, then the remaining $N - m$ signatures in the file could be skipped.

A number of signature clustering techniques have been proposed in the literature. They can be divided into two types: static where the signature space is prepartitioned into a fixed number of clusters; and dynamic where the number of clusters evolves as the file grows. Clustering methods may be further classified into those that require all the signatures before the clusters can be formed and those that create new clusters on-the-fly in response to insertion activity.

Lee and Leng [LEE89] and the DCC algorithm [FREN85] are examples of static clustering algorithms. Chang, et al. [CHAN89] and the convex clustering algorithm of Pfaltz [PFAL83] are dynamic and provide for the creation of clusters on-the-fly. The technique described in this paper also falls into the dynamic category.

Signature clustering will not affect the retrieval effectiveness of a system but it can drastically improve the efficiency. Effectiveness is decided at the time a document is mapped to a signature. This is essentially a labeling or classification scheme in which each document is assigned to one of the 2^L equivalence classes possible with an L-bit signature. Thus, signature formation is equivalent to document clustering if we regard the assignment of a signature as placing the document into the cluster labeled by that signature. With this in mind, it is clear that retrieval effectiveness will not be improved or degraded by signature clustering. With this interpretation, we may regard signature clustering as an effort to collect like document clusters together for the purpose of minimizing secondary storage access and time to process the signature file.

The following terminology will be used in the remainder of the paper. A *cluster* is a collection of signatures considered similar under some metric. The *cluster representative* is a signature formed by superimposing (inclusive OR) all the signatures in a cluster. The representative functions as a proxy for

the cluster members in subsequent retrievals. The *weight* of a signature refers to the number of bits set to one.

Two principles guiding the cluster formation process are:

1. The representatives of the forming clusters should exhibit slow weight growth; and
2. Consistent with (1) there should be a relatively small number of clusters with many members.

While principle (1) is self-evident, principle (2) may need some explaining. The motivation for clustering signature files is to reduce the search space on subsequent retrievals. Principle (2) simply means that if two different partitions can be formed by the clustering process with representatives having the same weight, then the partition having fewer subsets (clusters) should be preferred. In this way, the search space can be reduced more rapidly by pruning larger chunks.

The yardstick for a clustering algorithm should therefore be the rate of growth of the cluster representative weight as the number of members in a cluster increases. The slower the rate, the better trade-off we can get. The algorithm described in this paper seeks to achieve this goal.

4. Approaches to Clustering

Classic deterministic clustering techniques such as those used in data analysis [JAIN88] do not readily lend themselves to the clustering of signature files for two reasons. Firstly, they require that all signatures be present at the time of clustering. This is clearly impractical in dynamic office environments where documents and their signatures will be generated continuously. Secondly, these methods are computationally intensive and for very large file sizes the machine time required for processing will be too great.[1]

The clustering algorithm of Fig. 2 is typical of heuristic approaches to clustering where a set of clusters is dynamically evolving. The operation of the algorithm is straightforward. Each time a signature *S*

[1] Using complete-linkage, an $O(n^3)$ algorithm, it took an hour on an IBM RS6000 (during off-hours and low machine load) to get the dendrogram corresponding to 1000 16-bit signatures. It would take an enormous amount of processing time to cluster say 100,000 512-bit signatures.

The Clustering Algorithm

```
Initial values:
    n = 0
    best_similarity = −∞

    while S = get_signature()
        insert(S)


insert(S):

    for k = 1 to n do
        similarity = sim(S, R_k)
        if similarity > best_similarity
            best_similarity = similarity
            best_cluster = k

    if best_similarity > threshold
        C_best_cluster = C_best_cluster ∪ {S}
        R_best_cluster = R_best_cluster ∨ S
    else
        n = n + 1
        C_n = {S}
```

Figure 2

is considered for insertion in the clustered signature file, its similarity to each existing cluster representative R_k is evaluated using some function ($sim(S, R_k)$ in the figure) defined on these signatures. The cluster C_k having the greatest[2] similarity is then considered. If S is sufficiently similar to its representative R_k, then S is added to the cluster C_k otherwise a new singleton cluster is created. The variable *threshold* is a parameter of the algorithm which is used to decide when S is sufficiently similar.

Although for the sake of clarity of exposition we have assumed a linear structure, it is straightforward to extend this algorithm to hierarchical structures such as S-trees [DEPP86] and other tree structures

[2]If several clusters evaluate to the same similarity, the first one encountered by the algorithm is used. Other tie-breaking rules could be used.

[FREN85].

The crucial step of the clustering algorithm is the decision as to whether the incoming signature can be inserted into an existing cluster or a new one has to be created. This is called the decision step. The way this decision is taken leads to different algorithms in the category of heuristic clustering methods. A number of heuristics for the decision step have been considered [DEPP86, FREN85]. We describe below a heuristic decision rule which exhibits excellent behavior with respect to the goals outlined earlier for signature clustering.

5. The ODP Decision Rule

The motivation for this method comes from practices in statistical hypothesis testing in which one tries to deduce whether an event is sufficiently different from what one might expect to occur due to chance alone. This inspired the ODP (Overlap Driven Partitioning) [DAS92] rule for the decision step which is described next.

Consider a signature S with weight $weight(S)$. If we assume that any bit in a signature is equally likely to be set, the probability that any bit of S is set is:

$$\frac{weight(S)}{L}$$

where L is the length of the signature.

Let us consider two signatures A and B with known weights $weight(A)$ and $weight(B)$. The probability that a bit in any given position i is set in both A and B, assuming statistical independence, is the product of the two probabilities of it being set in each of them individually. So the expected number of bit positions which are set in both A and B or expected overlap (EOV) is given by

$$EOV(A,B) = \frac{weight(A)}{L} \frac{weight(B)}{L} L = \frac{weight(A)\,weight(B)}{L} \quad (1)$$

From these considerations we formulate the decision rule as: if A and B have an actual overlap sufficiently greater than the expected overlap for signatures of their respective weights, then put them in the same cluster. Hence the positive range of the measure

$$\Delta_{overlap} = \text{actual overlap} - \text{expected overlap} = weight(A \wedge B) - EOV(A, B) \quad (2)$$

provides a range of possible *threshold* values for the decision step.

To illustrate these concepts consider two 16-bit signatures A and B with $weight(A) = 12$ and $weight(B) = 8$. Then, the expected overlap of A and B from (1) is 6. The quantity $\Delta_{overlap}$ can be anything from -2 (when the overlap is the minimum i.e., 4 bits) to 2 (when the overlap is maximum i.e., 8 bits).

In the remainder of the paper we will use ODP to refer to the clustering algorithm of Fig. 2 with $sim(S, R_k) = \Delta_{overlap}$. Thus, when ODP receives an incoming signature, it calculates $\Delta_{overlap}$ with each of the existing cluster representatives. The decision step requires that for the incoming signature to be inserted into one of the existing clusters, max $\Delta_{overlap}$ has to be greater than t, a value given for the threshold parameter. If so, then the incoming signature is inserted into any one of the clusters whose representatives yielded a $\Delta_{overlap}$ value of max $\Delta_{overlap}$, with the tie-breaking rule being arbitrary selection of one of them (in Fig. 2 the first such cluster is chosen). If not, that is, if max $\Delta_{overlap}$ is less than or equal to t, a new cluster is created with the incoming signature as its sole member. This process repeats for each new insertion.

6. Properties and Behavior of ODP

Let us define the *stability* of a clustering method to be the degree to which the cluster characteristics obtained from different random samples agree with each other over a range of parameters, especially signature file size. The stability of ODP is demonstrated in Fig. 3 through Fig. 5. All the plots refer to an experiment done by taking 4 random sample streams of 32-bit signatures. Each signature had 16 bits set and was drawn at random from the $\binom{32}{16}$ possible signatures with this weight. The 4 sample streams were input to ODP, with a threshold of $t = 2$. The three quantities measured with increasing sample size for each of the 4 streams were the average number of members per cluster (Fig. 3), the number of clusters (Fig. 4), and the average weight of the cluster representatives (Fig. 5). As can be seen, the 4 sample

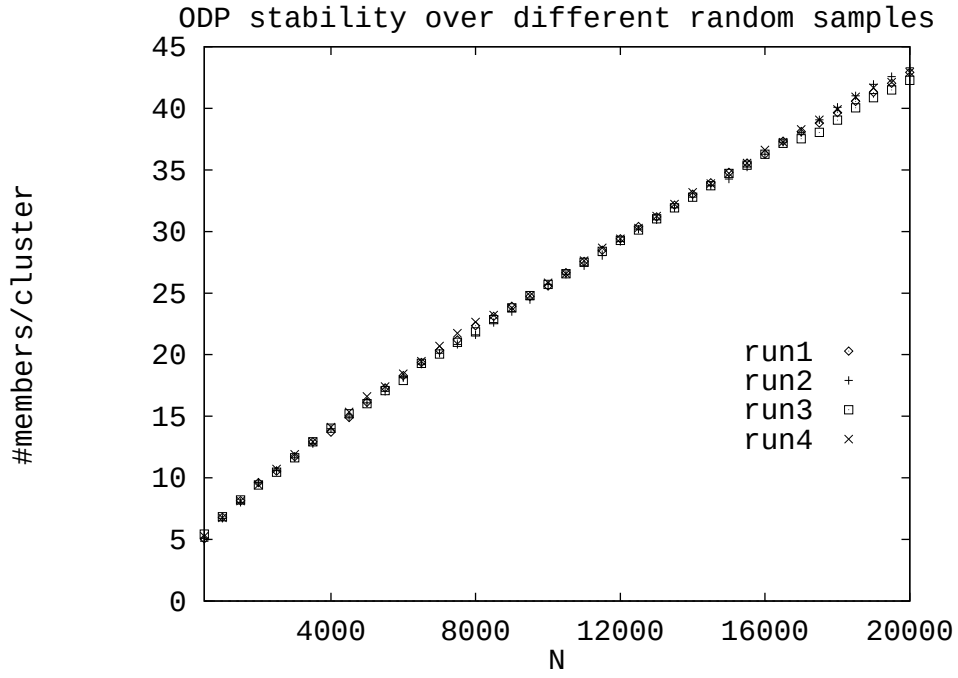

Figure 3

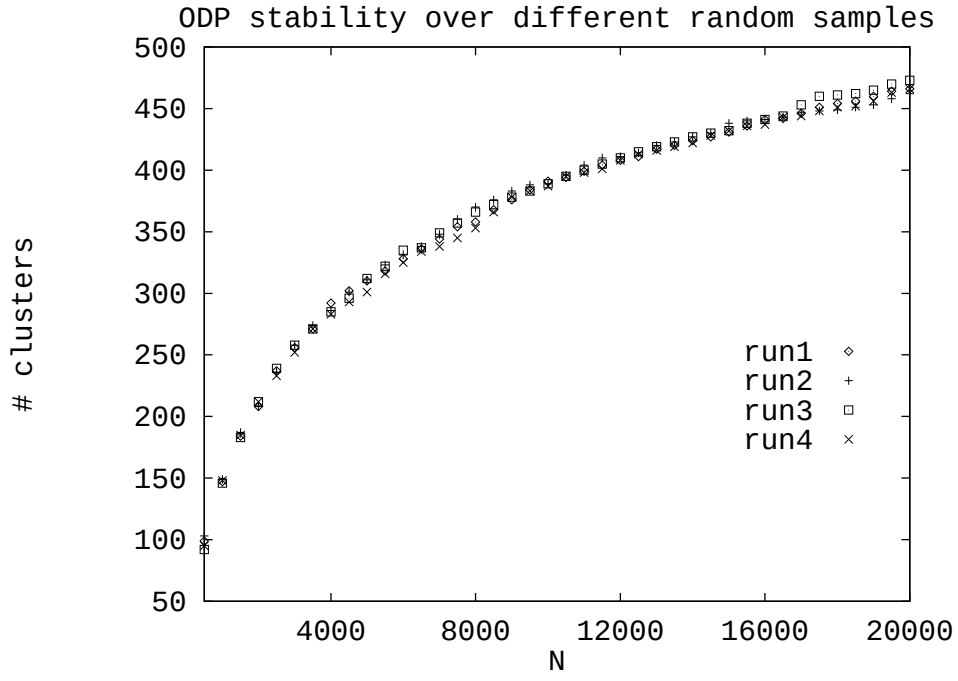

Figure 4

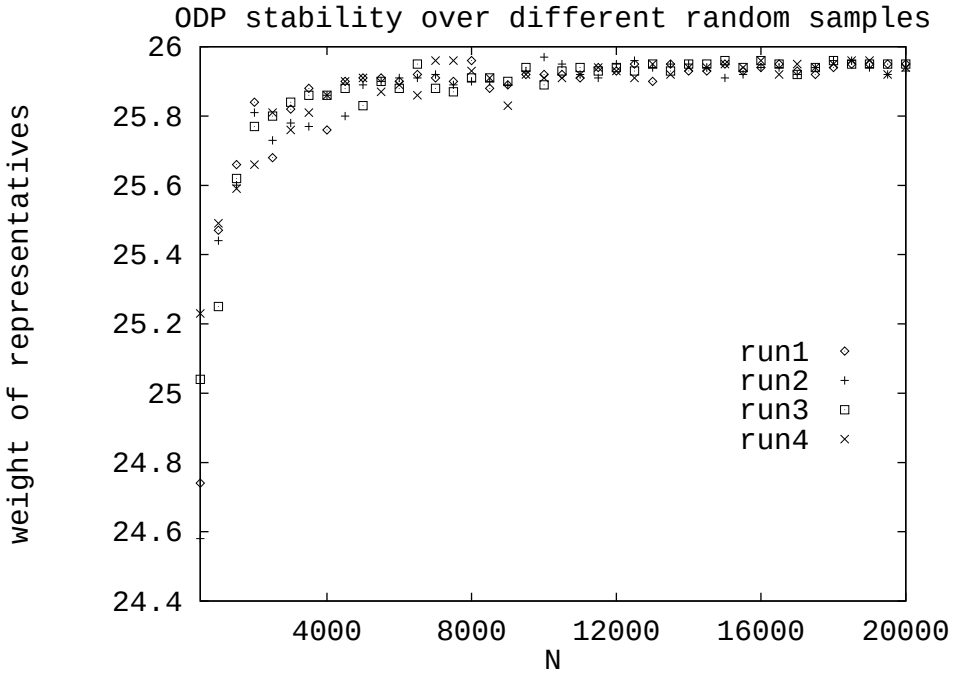

Figure 5

streams yield almost identical cluster characteristics.

The figures demonstrate much more than just the stability of the method. Recall that the desirable properties of clusters are slow weight growth in the representatives and high density in the clusters, resulting in fewer clusters. As the figures indicate, ODP clusters do have these properties.

Fig. 3 shows that the number of members per cluster is growing approximately linearly with file size. From Fig. 4 it appears that the number of clusters increases rapidly initially and then approaches a limit. This suggests that clusters are created more rapidly while the signature space is being covered by appropriate representatives. After the space is sufficiently covered, there is almost always a suitable cluster for every new signature and the rate of growth slows down. If the capacity of a cluster is left unbounded, eventually a point will be reached when no new clusters will be created and every new signature can be added to an existing cluster.

Fig. 5 shows that the weight of a cluster representative rapidly approaches a limiting value. The precise value is controlled by the signature length L and the threshold parameter t. For signatures having a constant weight (as has been assumed throughout the experiments reported here), it can be shown that

the weight of a cluster representative R_k of a cluster C_k is bounded by

$$weight(R_k) \leq L - 2(t+1) \tag{3}$$

In Fig. 5, $L = 32$ and $t = 2$ so that $weight(R_k) \leq 26$.

7. Comparing ODP with an Optimal Algorithm

A natural question to ask when evaluating the behavior of ODP (or any other heuristic clustering algorithm) is how well does it perform as compared to an optimal algorithm. The difficulty in doing such a comparison is getting the optimal set of clusters for a given signature file and threshold weight for the cluster representatives. There is no general algorithm other than exploring all possible partitionings. But, this is not computationally feasible because of the combinatorial explosion in the number of possibilities. The method we employ postulates the existence of an optimal algorithm and generates the clustered file that would have been produced by it. We then take the signatures in this file and cluster them with ODP and then compare the resulting clustered file with the postulated optimal file.

Our methodology is as follows. We consider signatures of length L and constant weight S. Let the maximum weight for any cluster representative be W. No cluster representative may exceed this weight. We wish to generate a clustered signature file with the following properties: (a) each cluster representative is exactly weight-W; (b) each cluster has exactly the same number of members; (c) no duplicate signatures exist; and (d) the number of clusters is fewest from among all partitionings. By *optimal clustered signature file*, we mean a clustered signature file satisfying these four properties. A constructive procedure for producing an optimal clustered signature file of weight-S signatures into weight-W clusters is outlined below.

Consider the set of all $\binom{L}{W}$ possible weight-W signatures. Let the set A be the largest subset of these weight-W signatures with the following property

$$A = \{ x \mid (\forall y \in A)\ y \neq x \rightarrow |\, x \wedge y \,| < S \}$$

Characteristics of Optimally Clustered Signature Files			
Weight of Representative	Number of Clusters	Members per Cluster	Total Signatures
9	715	9	6435
10	53	45	2385
11	6	165	990
Signature Parameters Length $L = 16$		Weight $S = 8$	

Table 1

By construction the set A is the largest set of weight-W signatures which, taken pairwise, do not overlap in more than $S-1$ bits.

Suppose we now take each member of set A and clear $W - S$ of the W set bits, in all possible ways. By this procedure we will generate all $\binom{W}{S}$ weight-S signatures from every weight-W cluster representative. If we take all these weight-S signatures and place them into a new set B, B will be a signature file for which A is an optimal set of cluster representatives. There cannot exist another set C of weight-W signatures with a cardinality less than A which can act as a set of cluster representatives for the signature file B. Note also that the condition that no two cluster representatives overlap in more that $S - 1$ bits guarantees that the same weight-S signature cannot belong to two clusters. Thus, this procedure will result in a clustered signature file satisfying the properties outlined above.

We generated a set A with the properties described and then a set B for which it acts as an optimal set of cluster representatives. We considered values of $W = 9, 10, 11$ and a constant weight of $S = 8$ for signatures of length $L = 16$.[3] We then used ODP to cluster the signatures in the set B for each of the three values of W to see how many clusters were generated. From (3), the corresponding values of the thres-

[3]These parameter values are obviously too small for actual signature files. They were chosen to drive the simulations quickly into regions where the asymptotic behavior would be observable. In the next section we discuss performance using more realistic parameter values, i.e., 100,000 512-bit signatures.

hold parameter t for ODP must be 2.5, 2 and 1.5 to enforce the weight bounds of 9, 10, and 11 respectively. The members of the set B were input to ODP in several different orders to capture the average behavior of ODP. The parameters used in the tests and the characteristics of the resulting signature files generated by this procedure are summarized in Table 1.

The results of the ODP tests are shown in Table 2. We could not go beyond $W = 11$ in the tests because for $W \geq 12$ it is not possible to construct the set A with more than one member in it if $S = 8$. For the case marked ‘‘opt’’, the signatures were presented to ODP in the order in which they were generated by the procedure described above. In all cases when ODP was presented the members of B in their order of generation, it produced the same number of clusters as the optimal.

ODP vs. Optimal Clustering

Cases	Number of Clusters	Average Weight
opt	715	9.00
1	1573	8.96
2	1587	8.96
3	1589	8.95
4	1590	8.96

(a)

opt	53	10.00
1	88	9.75
2	113	9.66
3	94	9.56
4	97	9.63

(b)

opt	6	11.00
1	13	10.62
2	9	10.44
3	14	10.57
4	7	10.71

(c)

Table 2

Cases 1 - 4 show the outcome when the generated sequence of signatures is arbitrarily permuted. This shows the sensitivity of the algorithm to the order in which the signatures are processed. For these cases, ODP produced roughly twice the number of clusters as in the optimal solution. The larger variability seen in Table 2(c) occurs because the number of members in set A is quite small.

We also used a deterministic clustering algorithm[4] on the files specified by the parameters of Table 1 and this algorithm produced the optimal clustered files. However, the running time of the deterministic algorithm was two orders of magnitude slower than ODP.

8. Performance of ODP

In this section we give some insight into the absolute behavior of files clustered using the ODP algorithm. We begin by developing a cost model to evaluate performance. The following derivation of a cost expression assumes that a signature file has been fully clustered and the clusters have a representative with which the query can be compared to find out whether or not any signatures matching the query can be found in them. This is thus a two-level index to the actual items which are encoded in terms of signatures. The first level is the signature file. The second level is the set of cluster representatives.

Let P be the number of clusters. Let m be the quantity given by the following equation

$$m = \left(\frac{\overline{w}}{L} \right)^{w_q}$$

where $\overline{w}$ is the average weight of a a cluster representative, w_q is the weight of a query and L is the length of a signature. The quantity m represents the probability that a query is subsumed by a cluster representative. Alternatively, it is the probability that the corresponding cluster is activated and its members are searched. Let b be the average number of members per cluster. Let B be the number of signatures which fit into a disk block. Let c_1 be the cost of comparing two signatures each of length equal to one bit. Let c_2 be the cost of a disk access. Then the generalized cost C for processing a query is given by

[4] A complete linkage algorithm using the Jaccard coefficient as the similarity measure was used [JAIN88, SALT88].

$$C = PLc_1 + mPc_2\left\lceil \frac{b}{B} \right\rceil + mPbLc_1 \qquad (4)$$

Equation (4) above has three terms. The first term denotes the cost of comparing signatures of the cluster representatives. The second term denotes the cost of the disk accesses that have to be made because mP is the expected number of cluster representatives that will match the query and $\left\lceil \frac{b}{B} \right\rceil$ represents the disk accesses per cluster. Similarly the third term denotes the cost of comparing signatures in the clusters that were fetched from disk since mP is the expected number of clusters activated, b is the average number of signatures per cluster and Lc_1 is the cost of comparing two signatures. Notice that we are assuming the P cluster representatives are in memory and hence no disk accesses are required to get them. Simulation showed that the value of P grows extremely slowly for increasing values of N. We have determined that for values of N and t of interest here there was never a memory requirement of more than 500K bytes or approximately 0.5M bytes of main memory for storing the cluster representatives. Since such memory requirements are easily satisfied on current machines including multiuser workstations the assumption seems reasonable. The cost equation given above can be rewritten as

$$C = \left(PL + mPK\left\lceil \frac{b}{B} \right\rceil + mPbL \right) c_1 \qquad (5)$$

In (5), the cost of disk access has been replaced by a multiple of the cost for comparing two 1-bit signatures. We can say that $c_1 = c_0/8$ where c_0 is the cost of comparing two bytes which usually take a few machine instructions. Today's state-of-the-art machines take about 40 nanoseconds for an instruction (25 MHz. clock cycle). Disk accesses in state-of-the-art disks take around 10 milliseconds for random access (not burst mode which is much faster since it does not include disk head repositioning) of standard block sizes like 4Kbytes. Even if one conservatively estimates ≈25 machine instructions for comparing two bytes (which should amply cover any overhead) and 1 msec for fetching a 4K disk block in a hypothetical fast disk, the constant K turns out to be 8×10^3. This value of K is appropriate for current technology; a different value could always be used if deemed suitable. We now have an usable equation

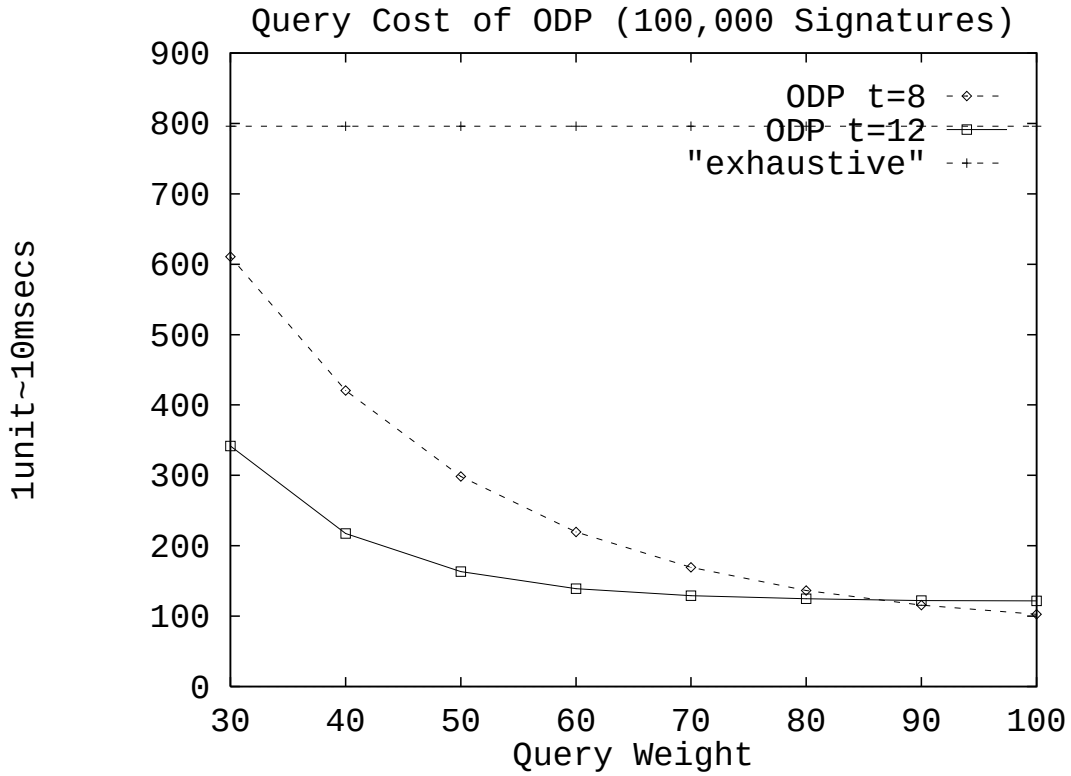

Figure 6

for our cost model, equation (5), which we shall refer to as the cost equation.

To get a feel for the retrieval performance, ODP was used to cluster a file of 100,000 512-bit signatures at threshold values of $t = 8$ and $t = 12$. The plots of (5) for these files are given in Fig. 6. A cost unit of 10 msecs is used on the y-axis and query weight on the x-axis. The value of B, the block size used, is 4K. The horizontal line at $y = 796$ in Fig. 6 represents the cost of an exhaustive sequential search as computed by the cost equation. In exhaustive search all signatures are searched for the query, that is, $m = 1$, $P = 1$, $b = N$ and furthermore, if it is assumed that all signatures are stored contiguously in secondary memory, then the number of disk accesses made is $\left\lceil \frac{N}{B} \right\rceil$. The cost equation then takes the form

$$C = \left(K \left\lceil \frac{N}{B} \right\rceil + NL \right) c_1 \tag{6}$$

The contiguous memory assumption made to arrive at equation (6) is not always true in practice. Any other arrangement increases the cost of disk accesses for exhaustive search. Therefore this is an optimistic estimate of the cost of exhaustive search. Even with this optimistic estimate, ODP shows very favorable performance improvements — almost an order of magnitude for query weights greater than 80.

9. Summary and Conclusions.

This paper discussed a new approach to clustering signature files which provides excellent retrieval performance and low insertion cost. This was established mainly from empirical arguments. The ODP algorithm is one of a class of weight limiting heuristics that we have been investigating. We expect to provide a more complete analysis of the behavior of these algorithms in the future. However, all empirical evidence available suggests that the ODP algorithm is an efficient mechanism for managing the storage and retrieval of text in very dynamic environments.

We also introduced a new methodology for evaluating signature clustering algorithms. The technique developed for constructing optimal clustered signature files can be useful in providing a common set of benchmark standards. This will make it easier to compare the performance of different clustering algorithms.

Acknowledgements. I wish to thank Mriganka Das who programmed the algorithm and collected most of the data discussed here.

References:

[CAN90] F. Can and E. A. Ozkarahan, ‘‘Concepts and Effectiveness of the Cover-Coefficient-Based Clustering Methodology for Text Databases’’, *ACM Trans. Database Systems 15*, 4 (Dec. 1990), 483-517.

[CHAN89] J. W. Chang, J. H. Lee and Y. J. Lee, ‘‘Multikey Access Methods Based on Term Discrimination and Signature Clustering’’, *ACM Transactions on Office Information System*, 1989, 176-185.

[DAS92] M. Das, ‘‘A Study of Some Algorithms for Partitioning Signature Files’’, Master’s Thesis, Dept. of Computer Science, University of Virginia, 1992.

[DEPP86] U. Deppisch, ‘‘S-tree: A Dynamic Balanced Signature Index for Office Retrieval’’, *Proc. ACM Conf. on Research and Development in Information Retrieval*, 1986, 77-87.

[FALO85] C. Faloutsos, ‘‘Access Methods for Text’’, *Computing Surveys 17*, 1 (Mar. 1985), 49-74.

[FREN85] J. C. French, *IDAM File Organizations*, UMI Research Press, Ann Arbor, MI, 1985.

[JAIN88] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[KENT88] A. Kent, R. Sacks-Davis and K. Ramamohanarao, ‘‘A Superimposed Coding Scheme Based on Multiple Block Descriptor Files for Indexing Very Large Data Bases’’, *Proc. 14th Inter. Conf. on Very Large Data Bases*, 1988, 351-359.

[KOTA90] U. Kotamarti and A. L. Tharp, ‘‘Accelerating Text Searching through Signature Trees’’, *Journal of the American Society for Information Science 41*, 2 (1990), 79-86.

[LEE89] D. L. Lee and C. Leng, ‘‘Partitioned Signature Files: Design Issues and Performance Evaluation’’, *ACM Trans. on Office Information Systems 7*, 2 (Apr. 1989), 158-180.

[LEVI91] J. Levitt, ‘‘The Promise of the WAIS Protocol’’, *UNIX Today*, Dec. 9, 1991, 44-48.

[PFAL80] J. L. Pfaltz, W. J. Berman and E. M. Cagley, ‘‘Partial-Match Retrieval Using Indexed Descriptor Files’’, *Comm. ACM 23*, 9 (Sep. 1980), 522-528.

[PFAL83] J. L. Pfaltz, ‘‘Convex Clusters in a Discrete m-Dimensional Space’’, *SIAM Jour. Computing 12*, 4 (Nov. 1983), 734-750.

[ROBE79] C. S. Roberts, ‘‘Partial-Match Retrieval via the Method of Superimposed Codes’’, *Proc. IEEE 67*, 12 (Dec. 1979), 1624-1642.

[SACK83] R. Sacks-Davis and K. Ramamohanarao, ‘‘A Two Level Superimposed Coding Scheme for Partial Match Retrieval’’, *Inform. Systems 8*, 4 (1983), 273-280.

[SALT78] G. Salton and A. Wong, ‘‘Generation and Search of Clustered Files’’, *ACM Trans. Database Systems 3*, 4 (Dec. 1978), 321-346.

[SALT88] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, MA, 1988.

[WILL88] P. Willett, ‘‘Recent Trends in Hierarchical Document Clustering: A Critical Review’’, *Information Processing & Management 24*, 5 (1988), 577-597.