AN ANALYSIS OF FILE SPACE PROPERTIES USING CLUSTERING

Barry G.T. Lowden and Jerome Robinson
Department of Computer Science, The University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, Essex,
United Kingdom
Telephone 0044 1206 872773 Fax 0044 1206 872788 Email lowdb@essex.ac.uk

Abstract

Clustering is a technique used in information retrieval to improve both search quality and response time. This latter is achieved through the placing of similar documents/records close together in the file space which in turn has the effect of reducing the total number of block transfers needed to retrieve a given record set. A file space which has been restructured in this way may be regarded as having a high *space density* relative to one in which the records are randomly distributed, space density being defined as a file property relating to the measure of record similarity or closeness.
In this paper we introduce a new clustering method which permits a file's space density to be varied and also show how this influences retrieval performance. The method is particularly suited to the restructuring of multi-attribute files of the kind found in conventional databases.
Keyterms : Information Retrieval, Database systems, Clustering

1. Introduction

In general most clustering strategies, described in the literature, are associated with document retrieval in library information systems [1], however the process may also be used to improve retrieval performance in conventional databases. Typically, a collection of records that is known to possess common characteristics may be organised (clustered) in such a manner that records of a particular grouping exhibit a given degree of similarity to one another. The result is a file in which sufficiently related records are placed in the same cluster [2, 3].

Given a query request, the database query processing software generates a set of addresses indicating where the records to be retrieved are stored. Since a memory block or page is the unit of both data allocation and transfer, retrieval of an individual record requires the transfer of the complete block in which it is located. Clustering then can lead to a reduction in the number of file accesses needed since there is a greater probability that a subset of the query result will lie in a single page of secondary storage and that the complete result set will be located within a minimal number of blocks. In addition to improving response time when accessing individual relations, clustering may also be used to bring together tuples from different relations, into common blocks, in order to increase the efficiency of join executions.

In this paper, we first review the criteria by which we may measure record similarity and then introduce a novel approach which enables the user to control the degree of file clustering. An analysis, of the results of clustering, is also presented which helps determine the tradeoff between the amount of clustering effort needed and the resulting improvement in retrieval performance. We then compare this new approach with a conventional clustering technique which is used as a benchmark.

2. Data Representation and Record Proximity Criteria

Let each record in a set of *N* records be represented by a set of *n* attributes, then each record may be considered as an ordered *n-tuple* of *n* values and is referred to as a pattern. Record characteristics may therefore represented in the form of vector variables in a multidimensional vector space. The file itself may be visualized as an *[Nxn]* pattern matrix where the rows of such a matrix define a vector or record and the columns define a feature or attribute. The most common method of measuring record similarity is based on the definition of indices of proximity between pairs of patterns (records) where the difference between the ***i***th and ***k***th pattern is denoted:

$$d(i,k)$$

For a dissimilarity index, the more the *ith* and *kth* objects resemble one another the smaller is their dissimilarity value. Jain & Dubes [4] specify a number of properties that must be satisfied.

$$\begin{aligned}
&d(i,i) = 0 \\
&d(i,k) = d(k,i) \quad \forall\ (i,k) \\
&d(i,k) \geq 0 \quad \forall\ (i,k) \\
&d(i,k) = 0 \quad \text{only if } x_i = x_k \\
&d(i,k) \leq d(i,m) + d(m,k) \quad \forall\ (i,k,m)
\end{aligned}$$

A wide range of measures of dissimilarity are available [5] however, for the purposes of this paper, we use *Hamming Distance* (*hd*) which is simply the number of terms by which two patterns differ. The *hd* between two patterns i and k may therefore be defined as:

$$d(i,k) = \sum_{j=1}^{d} f(x_{ij}, x_{kj})$$

$$\text{where } f(x_{ij}, x_{kj}) = \varnothing \text{ if } x_{ij} = x_{kj}$$

$$\text{and } \quad f(x_{ij}, x_{kj}) = 1 \text{ if } x_{ij} \neq x_k$$

Take the general case of an unclustered n-tuple file in which the records are considered to be randomly distributed across the file space [6]. If x is the number of matching terms between two adjacent records then the expected *hd* may be obtained by summing the product of hamming distance (n - x) and its probability of occurring P(x), over all possible values of x.

$$(1)\ \ E(hd) = \sum_{x=0}^{n} (n-x) \times P(x)$$

The possible values of x range between 0 and n inclusive. The two extremes being x = 0, when the records have no matching terms in common, and x = n when the records are identical.
Assuming the range of possible values of each attribute be drawn from a finite domain of size p, it may be seen that P(x) follows the Binomial Distribution and, after expansion and factoring out, the above expression reduces to:

$$(2)\ \ E(hd) = n(1-1/p)$$

for a full derivation, the reader is referred to [7].

The above expression gives the expected average *hd* between pairs of adjacent records in a random file and correlates well with experimental analysis. At this stage we introduce the notion of file *space density (sd)* which is a measure of record closeness or similarity. Expected space density may be determined by replacing *(n – x)* by *x* in the above expression (1) which may be seen to reduce to:

$$(3)\ E(sd) = n/p$$

Retrieval Performance based on a Random File

Let the random file, under consideration, consist of N records and be partitioned into B blocks or pages. Assume further that a general query results in the retrieval of a set of R records.
The probability that a given record, in the retrieved set, is located in a given block is:

$$1/B$$

and the probability that it is not is:

$$1-1/B$$

The probability that none of the R records are located in a given block is:

$$(1-1/B)^R$$

and so the probability that at least one of the records is located in the given block is therefore:

$$1-(1-1/B)^R$$

Hence the average number of blocks accessed to retrieve R records is:

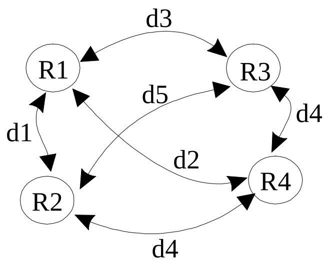
$$(4)\ \ B(1-(1-1/B)^R)$$

for a file with an expected average *hd* of *n(1-1/p)* as expressed in (1) above.

Clustering may be seen as an attempt to reduce the dissimilarity between records which, as we shall see, leads to improved retrieval performance [8]. In order to examine this relationship further, we now introduce a file restructuring algorithm which may be used to vary the record dissimilarity of a file according to a preset threshold.

3. Threshold Clustering

Given a file of *N* records and a dissimilarity index definition *d*, a complete weighted graph denoted by *(N,d)* can be constructed in which each of the N nodes represents a record in the file space and the weight on each one of the N(N-1)/2 edges of the graph represents the *hd* between the two records connected. An example is shown in Figure 1.



d1 = d[R1, R2]
d2 = d[R1, R4]
d3 = d[R1, R3]
d4 = d[R2, R4]
d5 = d[R2, R3]
d6 = d[R3, R4]

Figure 1. Example Graph

The algorithm proposed is based on constructing a spanning path through the nodes which reduces the sum of the *hd's* between them [9]. Records are inserted in turn, to an ordered set, at a position which only increases the file *hd* up to an amount pre-determined by a threshold value ***t***.

Algorithm:
A sub path of the graph *(N,d)* is an ordered subset of the nodes in *N*. A path *P* including an extra node *m* not in the original path is denoted by *(P,m)*. The path (P,m) may be constructed as follows:

CASE 1: If P passes through more than one node:
(a) find an edge (x,y) between the nodes of P such that the value
(d(x,m) + d(m,y)) - d(x,y)
is ≤ to some threshold value ***t*** (or is minimised if no such edge exists)
(b) delete edge (x,y) and add edges (x,m) and (m,y) to construct path (P,m)

CASE 2: If P passes through only a single node i then make path (p,m) the two node path consisting of edges (i,m) and (m,i)

Clearly the threshold value t will determine both the degree of clustering of the resultant file and also the effort needed to accomplish the restructuring. The range of t is 0 - n where *n* is the number of attributes in each record. If t = n then no clustering takes place. If $(t = 0)$ the resulting file is fully clustered and the work done is N(N+1)/2 comparisons.

The algorithm was further improved by adding a locking technique so that when running at threshold g, any adjacent pair of records with Hamming Distance < g would be considered as locked together. Treating sequences of locked records as extended records reduces the number of comparisons on the next and subsequent passes of the algorithm. Subsequent references to work done, in this paper, are based on the use of this locking technique.

Experimental Analysis

A series of model files was constructed consisting of a large number of fixed length multi-attribute records. The number of attributes per record could be varied as could the range of values which could be assigned to each attribute together with the file blocking factor. Attribute values were randomly distributed across the file space.

Results subsequently presented in this paper are based on running the algorithm, with different threshold values, against a model file of 10000 records each with six attributes having a range of different values. However it should be noted that these results are representative of a comprehensive analysis of several hundred runs using different model

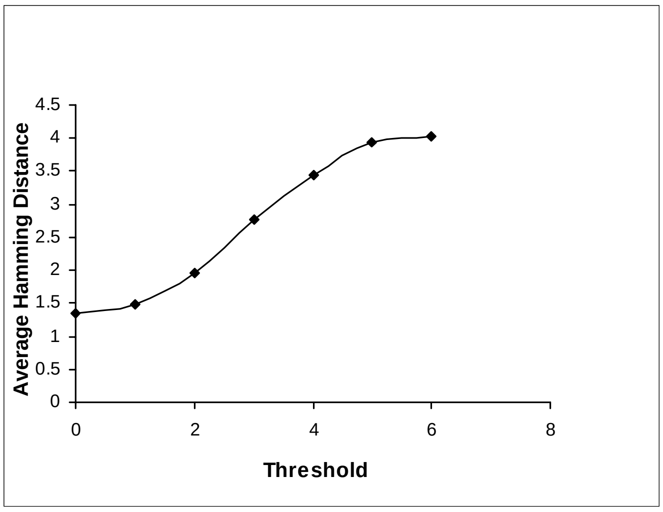


Figure 2. Hamming Distance against Threshold

files with varying parameters.

In Figure 2 we see how the average *hd* varies with the threshold chosen reflecting the degree of clustering achieved. Note that the upper value correlates well with the predicted value of n(1-1/p)= 4. In subsequent graphs we generalise the measurement of clustering by using the previously defined notion of space density instead of Hamming Distance. Space density is the average number of matching attribute values between adjacent pairs of records and will be expressed as a percentage. Clearly a value of 100% could only be achieved if all records in the file space were identical whereas a value of 0% would imply that all pairs of adjacent records were totally dissimilar.

Substituting for threshold values, the number of record comparisons needed to achieve that threshold, indicates a logarithmic relationship between space density and work done as shown in Figure 3. In this graph, the points represent threshold intervals.

To compute this relationship for a given file, two points of the graph may be determined, one of which is calculated from expression (4) above which is the density corresponding to a random file ie. zero work done. An approximation to a second point may be determined by applying the algorithm to a random sample of records from the file space. It was found that a record sample size of 5% over a wide range of model files (> 10K records) with varying characteristics, gave an acceptable approximation at t = n/2.

To summarise this section we may therefore, for a given file of records, determine the relationship between space density (degree of clustering) and related effort, to achieve that space density, by carrying out a relatively inexpensive experiment on a sample of the file records. In the next section we will examine how space density affects retrieval performance and how we can achieve significantly improved retrieval characteristics at low cost.

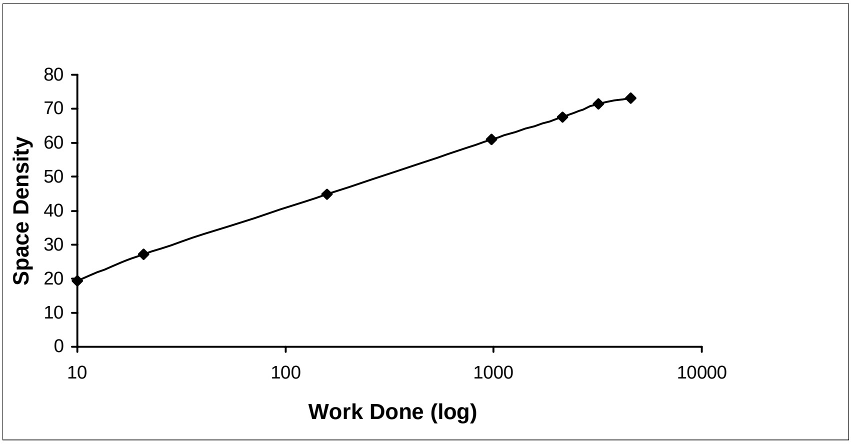


Figure 3. Space Density against Work Done (log)

4. Retrieval Performance

To evaluate retrieval performance, a random set of queries was generated containing varying numbers of terms. The query set was first run against the randomly distributed model file, the file was then clustered to different levels, using the threshold algorithm and the query set rerun against the structured file for each threshold value. As the space density value increases so does the probability that the retrieved set of records resides in a fewer number of blocks. Since retrieval time is directly related to the number of blocks accessed, retrieval performance should improve with increased space density.

The results are shown in Figure 4. As may be expected the blocks retrieved corresponding to the lowest space density (highest threshold) coincides with that for a random distribution and it may also be seen that retrieval performance then improves quite significantly with increasing density. This measure of improvement was typical over the full range of model files used and was near linear when the endpoints were ignored.

Since we have already seen how space density has a logarithmic relationship to work done, we may expect to see a similar relationship between work done and blocks retrieved. This is illustrated in Figure 5 with threshold intervals also shown.

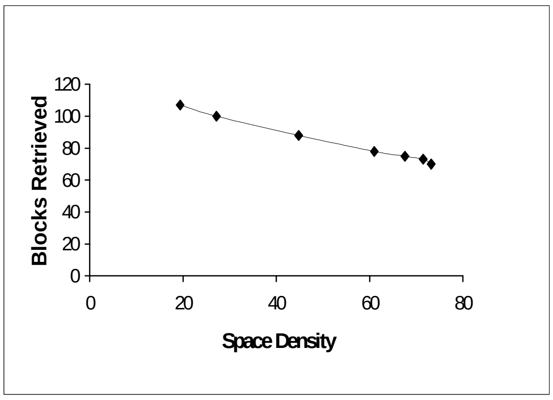


Figure 4. Blocks Retrieved against Space Density

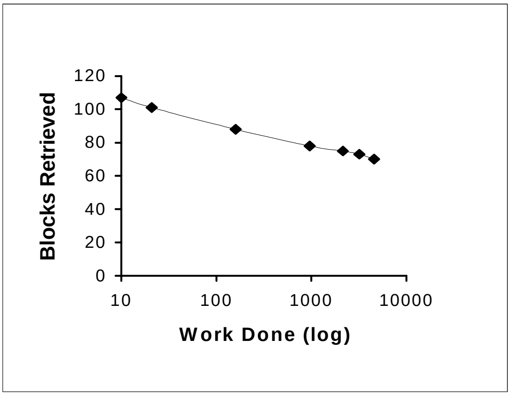


Figure 5. Blocks Retrieved against Work Done (log)

Again by estimating two points on the graph we may extrapolate to give a reasonable approximation of performance v work done for any given file, the gradient correlating with the specific physical characteristics of the file in question. This enables us to predict with reasonable accuracy how much effort should be expended to achieve required performance for a given file usage and volatility.

5. Comparison and Evaluation

The clustering algorithm described above permits the space density of a file to be varied using a threshold parameter. This flexibility is not generally a characteristic of other commonly used approaches whose complexity is normally a straightforward function of the number of records in the file space. Representative of the techniques applicable to a situation where the pattern of queries is unpredictable, is Johnston's (Single Link) algorithm.

In this algorithm, first suggested by King [10], all N records are initially considered to constitute N disjoint clusters, one record to each cluster. The only input to the algorithm is a proximity matrix, denoted by

$$D = [d(i, j)]$$

Each entry (i,j) in the matrix denotes the Hamming Distance between the corresponding clusters (r) and (s) and is denoted by

$$d[(r),(s)]$$

The algorithm begins by searching the disjoint clusters, represented by the *NxN* proximity matrix, for the least dissimilar pair of clusters. These clusters are then merged into a single cluster and the matrix modified accordingly. This process is repeated until only a single cluster is left.

The work done to construct the *NxN* proximity matrix, computed as the number of comparisons between two records, is $N^2/2$ [4]. Since the records are assumed to be randomly placed then the entire matrix must be searched each time.

The performance characteristics of the Single Link algorithm were evaluated using the same series of model files and query sets as in the earlier experiments. Since the degree of clustering in this algorithm cannot be varied, it is only possible to show the fixed performance improvement for different blocking factors (Figure 6).

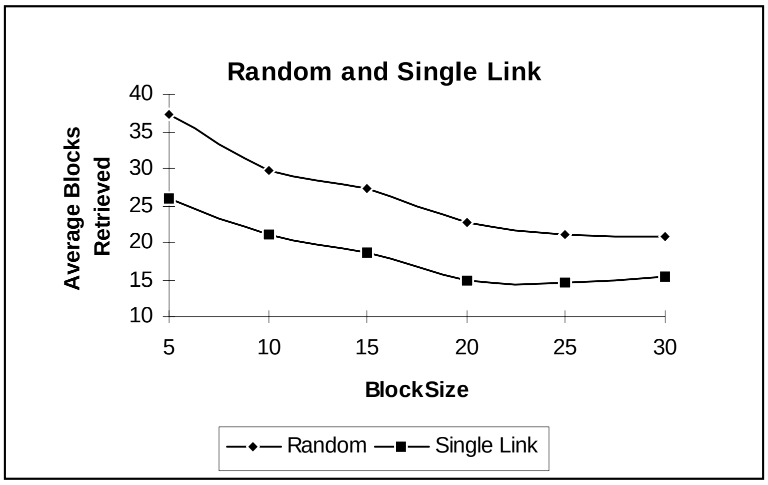


Figure 6. Performance Comparison

This improvement relates to the reduction in the number of blocks retrieved, in response to a random query, as a result of clustering a previously random file. The Single Link algorithm, however, consistently failed to achieve a space density equivalent to the maximum achieved by the threshold algorithm at t=0, and consequently yielded a smaller improvement in terms of reduction in blocks accessed.

Comparative results are shown in Figure 7 which uses Johnston's algorithm as a bench mark for the threshold algorithm. As the threshold value increases so the work done in restructuring the file decreases and this is shown, in brackets, as a percentage of that required with t=0, i.e. the same as Johnston's algorithm. From this graph, it may be seen that the threshold algorithm yields a performance equivalent to that of Johnston's algorithm at a threshold value of approximately $\lceil n/2 \rceil$ a result which was typical across a wide range of experiments.

Running the threshold algorithm at t = $\lceil n/2 \rceil$ provides a simple heuristic for achieving an acceptable performance improvement at little cost and may be preferred as a simpler alternative to the performance estimation method of the last section. Clearly, however, the estimation approach will give greater control over retrieval performance and precise measurement may be crucial in sensitive applications.

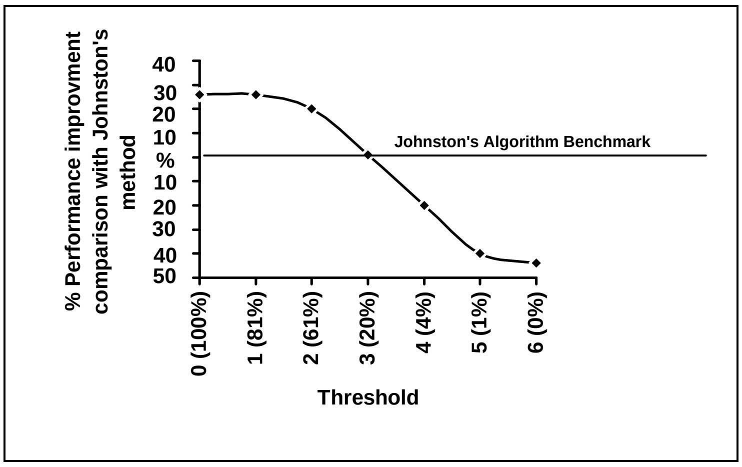


Figure 7. Threshold Algorithm Performance Compared with Benchmark

6. Summary

This paper has sought to analyse the relationship between file clustering and retrieval performance using the notion of space density. The ideas are applicable to files which have fixed length multi-attribute records and so are particularly suited to a commercial environment.

A threshold algorithm has been introduced which permits the space density of a file to be varied, by reducing the hamming distance between records, and experimental results presented showing how certain performance improvements may be achieved with different levels of clustering. This enables database designers to make decisions regarding the trade-off between performance and the file restructuring effort needed to yield that performance.

Finally the results have been compared with those yielded by a conventional single link clustering algorithm (Johnston's) using the latter as a benchmark. In all cases the threshold algorithm outperformed Johnston's algorithm and provided much greater flexibility in controlling performance/effort ratios.

7. Bibliography

[1] G.Salton. *Automatic Text Processing*, Addison-Wesley, 1989.

[2] C.T.Yu, Cheing-Mei Suen, K. Lam, and M.K. Siu, *Adaptive record clustering*. ACM Transactions on Data Base Systems, Vol 10, No 2, 1985.

[3] G Salton, J Allan & C Buckley, *Automatic structuring and retrieval of large text files*, CACM, 37(2), February 1994.

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

[5] G. Salton, M.J.McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill 1983.

[6] C.T. Yu, W.S.Luk and M.K.Siu. *On the estimation of the number of desired records with respect to a given query*. ACM Transactions on Data Base Systems, Vol. 3, No. 1, 1978.

[7] A. Kitsopanidis, *Analysing file space properties in a multiattribute environment*, MSc Dissertation, Computer Science, University of Essex, 1995.

[8] C.T. Yu & W.S. Luk, *Analysis of effectiveness of retrieval in clustered files*, JACM 24(4), October 1997.

[9] B.G.T Lowden, *An Approach to multikey sequencing in an equiprobable keyterm retrieval situation*, ACM SIGIR, 1985

[10] B.King, *Step-wise clustering procedures*, Journal of the American Statistical Association, 1967