ESTIMATING BLOCK ACCESSES WHEN ATTRIBUTES ARE CORRELATED

Brad T. Vander Zanden, Howard M. Taylor and Dina Bitton

Cornell University

Ithaca, NY 14850

ABSTRACT

Most database systems fallaciously assume that attributes are independent. This assumption leads such systems to systematically overestimate the costs of queries and thus to select execution strategies that substantially increase the queries' processing time. In this paper we show how the concepts of Schur concavity and majorization can be used to efficiently estimate the cost of a query when the queried attribute is correlated with the clustering attribute. We will also examine how a block access distribution can be constructed when attributes are correlated in this manner.

1. INTRODUCTION

Most database systems assume that attributes are independent. In fact, functional, multivalued, and implicit dependencies generate a great deal of dependence and correlation between attributes. This dependence has the effect of reducing the number of blocks that must be accessed to process a query. Thus many database systems overestimate the block cost of a query, causing them to select access strategies that could dramatically increase the cost of executing a query. Several recent studies have produced both theoretical and empirical results that confirm the pessimism of the independence assumption [Christodoulakis 1981 and 1984b; Montgomery et al. 1984].

Despite these shortcomings, most designers incorporate an independence assumption into their database models. By doing so they obtain simple expressions that rapidly estimate the costs of alternative query execution strategies. This simplicity is advantageous since the selection of a strategy is part of the overhead associated with a query. If the query optimizer cannot rapidly estimate the costs of alternative processing strategies, it might be able to execute the query more rapidly by randomly choosing an access strategy.

Unfortunately, realistically modeling the database environment leads to cost formulas that cannot be efficiently evaluated. Each block has a certain probability of storing a record that is requested by a query, and these probabilities are normally modeled by empirical block access distributions. More precisely, the block access distribution of a query is normally represented as a vector $p = \langle p_1,...,p_m \rangle$ where p_i denotes the probability that the query accesses the ith block and m denotes the number of blocks in the file. The time required to estimate the cost of a processing strategy is proportional to the number of distinct elements in this vector. Normally this number is approximately m. For example, Zahorjan and his associates [1983] employed a technique derived from the study of queuing networks to develop an O(km) time algorithm that computed the expected number of blocks accessed by a query (k = number of records requested and m = number of blocks in the file). Christodoulakis [1984a] took a somewhat broader view of the problem, attempting to both construct the vector **p** and then to derive a cost estimate based on this vector. He derived the block access distribution by treating the location of a record in secondary storage as an additional attribute value and then using a multivariate parametric distribution to describe the distribution of tuples in this "extended" relation. By integrating the probability density function over the appropriate ranges of attribute values, he was able to derive the block access distribution for a given query. He then developed an O(m) expression that computed the expected number of blocks accessed by a query. Several other estimation approaches that have been considered include simulation [Siler 1976] and analytical modeling [Demolombe 1980; Luk 1983; Christodoulakis 1983a].

The O(m) time that these studies require to estimate the cost of a query may preclude their use in query optimization or physical design settings that involve very large databases (since m is very large in this case). Thus some method must be found that reduces the number of distinct elements in the **p** vector. In addition, many of these studies do not address the problem of constructing the query's block access distribution when the queried and clustering attributes are correlated. Only Christodoulakis [1984a] discussed at any length the impact that this factor has on block access distributions and on the number of block accesses required to execute a query. However, he focused on situations where tractable parametric distributions, such as normal or Pearson distributions, apply. In many situations the underlying distribution is not known and nonparametric techniques must be brought to bear. The problem, then, is to construct an empirical distribution that 1) contains relatively few distinct elements and 2) accurately estimates the number of block accesses required by a query when correlated attributes are present.

We approach this problem in two steps. First, we use an occupancy model to derive an expression that calculates the expected number of blocks accessed by a query [Feller 1968; Kotz and Johnson 1977]. We then proceed to develop robust, accurate approximations for these formulas based on the dual

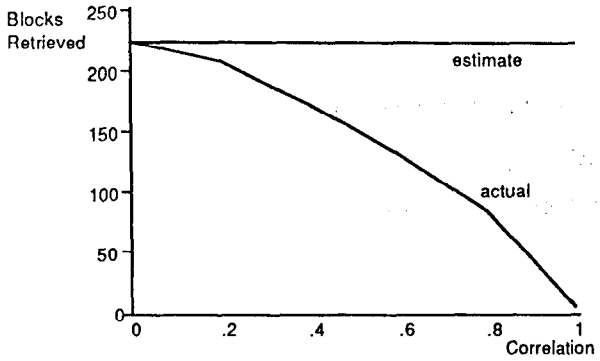


Figure 2.1 -- Comparison between the number of block accesses predicted when attributes are assumed independent and the actual number of block accesses required to process a query that retrieves 300 records. It is assumed that the correlation between the clustering attribute and the queried attribute varies between 0 and 1.

concepts of Schur concavity and majorization (see section 3). More specifically, the rest of the paper is organized as follows. In section 2 we present the results of a simulation that demonstrates the effect of correlation on the cost of a query and that pinpoints the shortcomings of the independence assumption. In section 3 we demonstrate how a compact block access distribution can be constructed that allows query optimizers to efficiently and accurately estimate the number of blocks that must be accessed to execute a query that retrieves k records. We assume that the queried and clustering attributes are correlated. In section 4 we examine the problem of constructing an empirical block access distribution and in section 5 we evaluate the performance of the estimates developed in sections 3 and 4 in the context of the simulation described in section 2. Finally in section 6 we summarize our results.

2. EXAMPLES

In this section we will analyze the results of several experiments that demonstrate the impact that correlated attributes have on the number of blocks accessed by a query. These experiments will also help us pinpoint situations in which the independence assumption generates block estimates that differ significantly from the actual number of blocks accessed by a query.

2.1. Experimental Design

In our experiments we built a 25,000 tuple relation R with two attributes A and B. The relation was divided into 500 blocks of 50 tuples each. Further, the attributes A and B were related by the multivalued dependency A ->-> B. The attribute domains of A and B both consisted of the integers {0,1,2,...,49,50} and the relation was clustered on attribute A. The elements for attribute A were normally distributed with mean 25 and standard deviation 10. That is, 68% of the tuples in relation R contained values for attribute A which were between 15 and 35, and 95% of the tuples contained values which were between 5 and 45. To simulate the multivalued dependency A ->-> B, we employed the formula t[B] = t[A] + 51c(U - 1/2). In this formula, t[A] and t[B] denote the values associated with attributes A and B in an arbitrary tuple of R, c varied between 0 and 1.8, and U was a uniformly distributed random variable on the interval [0,1]. If the value of B generated by the formula fell outside B's domain, the value was rejected and another value generated. This process was repeated until an acceptable value was obtained. The parameter c controlled the correlation between the attributes A and B with smaller values of c corresponding to greater correlation. This can be intuitively

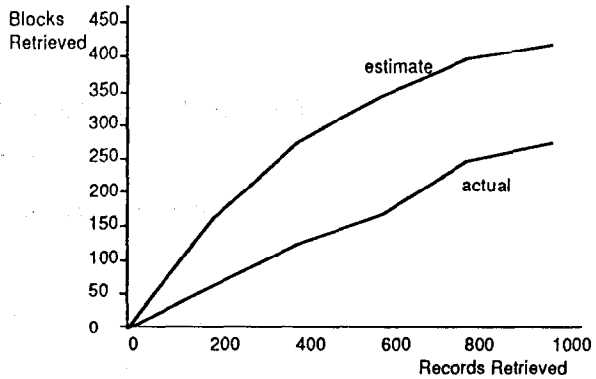


Figure 2.2 -- Comparison between the number of block accesses predicted when attributes are assumed independent and the actual number of block accesses required to process a query that retrieves a varying number of records. It is assumed that the correlation between the queried attribute and clustering attribute is .8.

inferred from the formula since small values of c force the B value to be clustered near the A value whereas larger values of c allow B to vary more widely. Our experiments showed that the correlation had an approximately negative, inverse relationship with c -- as c increased from 0 to 1.8, the correlation declined from 1 to 0. Negative correlation was not tested in our experiments since the results would mirror the results for positive correlation.

In our experiments we tested the query "Retrieve all tuples from relation R where R.B = constant" where the constant was chosen from the domain of attribute B. For each such value we recorded the number of blocks that contained the value and compared this number with the number predicted by the formula

$$E[X_k] = m\,[1 - (1 - 1/m)^k] \qquad (2.1)$$

This formula represents the estimated number of blocks accessed by a query that retrieves k tuples [Cardenas 1975]. It assumes that attributes are independent and that tuples are randomly placed in secondary memory. The formula is derived by noting that 1/m represents the probability that a block contains one of the requested tuples, 1 - 1/m represents the probability that a block does not contain one of the requested tuples, $(1 - 1/m)^K$ represents the probability that a block does not contain any of the k requested tuples, and $1 - (1 - 1/m)^K$ represents the probability that a block contains at least one of the k requested tuples. By summing this expression over all m blocks, we obtain expression (2.1).

2.2. Results

In figures 2.1 and 2.2 we have plotted some representative results from our experiments. Figure 2.1 illustrates how the number of block accesses required to retrieve 300 records declines as the correlation between attributes A and B increases from 0 to 1. It also demonstrates that as long as the correlation is less than 0.4, expression (2.1) provides a fairly reliable estimate for the actual number of blocks accessed. While the relative error of its estimates increases to approximately 30% for a correlation of 0.4, the estimates it provides would probably not lead a query optimizer to make a bad decision--the appropriate strategy seems to be one of scanning the entire relation and the optimizer will probably make this choice. As the correlation between attributes A and B increases beyond 0.4, the performance of expression (2.1) degrades badly. Its estimates deviate by more than 70% from the actual number of block accesses when the correlation is 0.6 and the error rapidly increases to more than 3000% as the correlation approaches 1. In addition, if the query optimizer could

accurately estimate the number of blocks accesses that are required when the correlation is 0.6 (approximately 135 blocks must be accessed), it would probably choose an index scan. However, the estimate generated by expression (2.1) would probably lead the optimizer to choose a strategy that scans the entire relation, a strategy that could substantially increase the query's processing time.

Figure 2.2 compares the actual number of block accesses required to retrieve a varying number of records with the number predicted by expression (2.1). It provides further evidence that the independence approximation produces unduly pessimistic estimates when the correlation between attributes A and B is somewhat high. The difference between the actual number of block accesses and the number estimated by expression (2.1) varies between 50% for 800 or more records retrieved to almost 400% for less than 100 records retrieved. The deviation between the actual number of block accesses and the estimates generated by expression (2.1) when relatively few records are retrieved is especially serious since the query optimizer will probably choose a relation scan rather than the cheaper index scan. For example, if a query retrieves 200 records, the index scan strategy could be as much as three times faster than the relation scan strategy (180 blocks (60 for the actual data + 120 index blocks) for an index scan versus 500 blocks for a relation scan).

What conclusions can be drawn from these experiments? They show that when the correlation between a clustering attribute and a queried attribute is less than 0.4, the independence assumption provides fairly reliable block estimates. However, if the correlation is moderately large (i.e., greater than 0.4), the independence assumption generates block estimates that badly mislead the query optimizer. These estimates may cause the optimizer to choose strategies that unnecessarily increase the time and effort required to process the query. Strong correlation between attributes often arises in database settings. Functional dependencies represent one extreme where clustering on one attribute imposes a good deal of order on other attributes. However, even weaker dependencies such as multivalued dependencies and implicit dependencies (e.g., between salary and position) induce strong correlation. Thus if a database designer suspects that two attributes may be strongly correlated, he would be well advised to drop the assumption of attribute independence and look for other ways to estimate the number of block accesses. In the next section, we will develop such a mechanism.

3. BLOCK ESTIMATE EXPRESSIONS

In this section we will show how rapidly computable yet accurate block estimates can be derived when queried and clustering attributes are correlated. For concreteness, we will consider queries that retrieve k records from a set of blocks labelled $B_1,B_2,...,B_m$ (we will call this query a type 1 query). Most queries can be reduced to this form by passing them through a module that estimates the record selectivity of their various operations. The work on estimating record selectivities has proceeded more rapidly than the work on estimating block selectivities and several excellent papers have been written on the subject [Merrett and Otoo 1979; Kerschberg et al. 1982; Christodoulakis 1983b; Piatetsky-Shapiro and Connell 1984; Kamel and King 1985].

3.1. Precise Block Estimates

An analytical estimate for the number of blocks accessed by a type 1 query can be derived by considering a related occupancy problem [Feller 1968; Kotz and Johnson 1977]. The advantage of placing our work in this context is that we are able to draw upon the many results obtained in this area by statisticians.

To transform our block estimate problem into an occupancy problem, we must first formalize it as follows: A query randomly retrieves k records from a file that is divided into m blocks and that contains n records. Let $\mathbf{p} = \langle p_1,...,p_m \rangle$ denote the block access distribution of this query where p_i denotes the probability that the query accesses the ith block. We assume a tuple may be retrieved more than once (this form of retrieval is termed sampling with replacement--although most queries retrieve records no more than once, Yao [1977] demonstrated that when the blocking factor is ten or greater, the results obtained for sampling with replacement are, for practical purposes, identical to those obtained for sampling without replacement). How many blocks must be retrieved?

An equivalent occupancy problem can be constructed as follows. A set of k balls is randomly assigned to a group of m urns that can contain a maximum of n balls. On each toss, the ith urn has a p_i probability of being assigned a ball. How many urns are occupied? By associating urns with blocks and balls with tuples we establish an equivalence between the block access problem and this occupancy problem.

Normally, a different number of urns will be occupied each time the experiment is performed. Thus we must find an approximate measure for the number of occupied urns. The measure most often used by computer scientists and statisticians is the *expected* number of occupied urns. The expression for the mean number of occupied urns is [see, for example, Kotz and Johnson 1977]

$$E[X_k] = \sum_{i=1}^{m} [1 - (1 - p_i)^k] \qquad (3.1)$$

where

X_k = number of occupied urns when k balls are thrown

$E[X_k]$ = expected number of occupied urns when k balls are thrown

$1 - p_i$ = probability that the i^{th} urn is not assigned a ball on each toss

$(1 - p_i)^k$ = probability that the i^{th} urn is not assigned a ball on any of the k tosses

$1 - (1 - p_i)^k$ = probability that the i^{th} urn is assigned at least one ball in k tosses

Christodoulakis independently derived this expression in the context of block accesses. When records are randomly assigned to blocks (i.e., $p_i = 1/m$), expression (3.1) reduces to expression (2.1).

The validity of this measure depends on the variability in the number of occupied urns that is observed in repeated experiments. The number of occupied urns is governed by a probability distribution, often called the occupancy distribution. Expression (3.1) represents the mean or expected value of this distribution. Statisticians have shown [see for example Kotz and Johnson 1977] that in many instances, this distribution is asympotically normally distributed. They have also shown that the standard deviation of this distribution is quite small relative to the mean. Thus expression (3.1) represents a good estimator of the number of blocks retrieved. Indeed, when each block has an equal probability of being accessed (i.e., $p_i = 1/m$), it can be

shown that asymptotically, both the mean and standard deviation grow linearly with the number of blocks. Thus, as the number of blocks in a file grows, expression (3.1) becomes an increasingly accurate estimator for the number of occupied urns, or equivalently, the number of accessed blocks.

3.2. Majorization

Although expression (3.1) provides accurate block estimates, the O(m) flops that are required to compute it makes it an impractical tool in many query optimization settings. In addition, for large files, it will be prohibitively expensive to maintain the empirical block access distribution. On the other hand, we have already noted that standard approximations such as expression (2.1) may produce highly distorted estimates when attributes are strongly correlated. Ideally, some sort of "compaction" algorithm should be applied to the block access distribution that would replace its original **p** vector with a vector that contains very few distinct values (i.e., a vector with many duplicate values). The catch is that the newly constructed vector must capture enough information from the original block access distribution to provide reliable block estimates. Fortunately, expression (3.1) possesses a property known as Schur concavity that aids us in this matter.

Definition. Let $\mathbf{y} = \langle y_1,...,y_m \rangle$ and $\mathbf{z} = \langle z1,...,z_m \rangle$ denote two vectors of nonnegative, nonincreasing (i.e., $y1 \geq y2 \ldots \geq y_{m-1} \geq y_m$ and $z_1 \geq z_2 \ldots \geq z_{m-1} \geq z_m$) real numbers. We say that vector **y** *majorizes* vector **z** (written **y** > **z**) if the following set of inequalities hold [Marshall and Olkin 1979]

$$\sum_{i=1}^{k} y_i \geq \sum_{i=1}^{k} z_i \text{ for all } k \leq m-1 \text{ and } \sum_{i=1}^{m} y_i \geq \sum_{i=1}^{m} z_i \text{ for } k = m$$

Majorization is a measure of non-uniformity in the sense that if **y** majorizes **z**, then the distribution represented by **y** is more diffuse (i.e., less uniform) than the distribution represented by **z**.

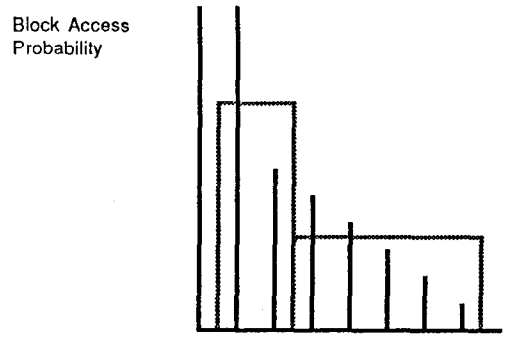
Definition. A real-valued function $f(x_1,...,x_m)$ defined over the set of non-negative real vectors R_+^m is said to be *Schur concave* if

$$\mathbf{y} > \mathbf{z} \Rightarrow f(\mathbf{z}) \geq f(\mathbf{y})$$

In other words, as the diffuseness of a vector of real numbers increases, the value of the function *f* decreases. A more practical way of determining whether a function is Schur concave is to apply the following test: a function $f(x_1,...,x_m)$ of m real variables is *Schur concave* if for every pair i,j, $(x_i - x_j)(\partial f/\partial x_i - \partial f/\partial x_j) \leq 0$ [Marshall and Olkin 1979]. Christodoulakis [1984b] employed this test to show that expression (3.1) is Schur concave.

In the context of the block estimate problem, the property of Schur concavity indicates that as the diffuseness of a distribution increases, the expected number of blocks accessed by a query decreases. As the correlation between two attributes increases, the block access distribution does become increasing diffuse. Since the independence assumption employs a vector that is majorized by the vector of the true access distribution, we can now understand why expression (2.1) often leads to pessimistic block estimates (the independence assumption uses the vector $\langle 1/m, 1/m,..., 1/m \rangle$).

The property of Schur concavity also guides us in finding a compaction algorithm for the block access distribution [Vander Zanden et al. 1985]. If we can construct a vector that contains very few distinct values but which is majorized by the original **p** vector, then the properties of Schur concavity and majorization guarantee that the block estimates it generates will provide an upper bound for expression (3.1).



Histogram approximation for the distribution of a sample **p** vector
Figure 3.1

The first step of the compaction algorithm involves permuting the elements of **p** so that they form a decreasing sequence $\mathbf{p} = \langle p_1',...,p_m' \rangle$. Since permutations of the elements in a vector do not affect the value of expression (3.1), this step does not alter the block estimates. The next step of the algorithm involves approximating the permuted block access distribution with an unequal interval histogram (see figure 3.1). The height of each box in the histogram is equal to the average block access probabilities of the elements that comprise it. In essence it is making the nonuniform distribution more uniform. Each box of the histogram divides the permuted **p** vector into subvectors. The elements within each subvector are replaced with a similar number of equal-sized elements whose block access probabilities are equal to the average block access probability of the original elements. The newly constructed subvectors are majorized by the original subvector since any vector of non-negative, nonincreasing reals majorizes a vector of equal-sized components (provided that the sum of the elements in each vector are identical). By concatenating these new subvectors, we obtain a vector that is majorized by **p** but which contains very few distinct elements. Thus an upper bound for expression (3.1) can be quickly computed by plugging in the newly constructed vector. The approximation can be written as follows

$$E[X_k] \approx \sum_{i=1}^{m'} u_i[1 - (1 - p_i'')^k] \qquad (3.2)$$

where m' = number of intervals in the histogram, u_i = number of elements in the ith interval of the histogram, and p_i'' = average block acess probability of the elements in the ith interval of the histogram.

This algorithm has the theoretically satisfying property that as the number of intervals is increased, the approximation becomes increasingly accurate. Expression (2.1) represents one extreme of this algorithm that uses only one interval. Similarly expression (3.1) represents the other extreme in that it uses *m* intervals. The algorithms we present strive to hold the number of intervals to 2-4.

So far we have avoided the issue of how the intervals of the histogram should be chosen. The selection of these intervals is critical since a poor partition can lead to an upper bound that is only slightly lower than expression (2.1). Several considerations must be taken into account in making this choice:

1. Homogeneous Values: Elements of the permuted vector **p** that generate nearly identical values when they are plugged into the expression $[1 - (1 - p_i)^k]$ should be grouped together. For

example, suppose the block access distribution is given by the vector **p** = <.4,.3,.1,.1,.05,.05>. Since the values .4 and .3 are likely to produce similar values they should be placed in the same interval. Similarly the values .1, .1, .05, and .05 should be grouped together.

2. Size of K: The best partition of the **p** vector will often depend on the size of k. As k increases, an increasing number of elements, p_i, will produce expressions $[1 - (1 - p_i)^k]$ that evaluate to 1. In other words, as k increases, an increasing number of blocks will be accessed with probability 1. Ideally, as k increases, the number of elements in the first interval of the histogram (see figure 3.1) should also increase.

The algorithm we present in section 3.3 for partitioning a vector addresses the issue of homogeneous values while in section 4.1 we present an algorithm for reconfiguring these partitions based on the size of k.

3.3. Mean Algorithm

The mean algorithm partitions the block access distribution based on one of its characteristics, the mean. The idea behind the scheme is that all elements less than the mean of a vector are grouped into a subvector and all elements greater than the mean of a vector are grouped into a subvector. This splitting can then be recursively applied to each of the subvectors. The algorithm is made precise in figure 3.2.

The restriction of the algorithm to intervals that are a power of two is not a significant hindrance in practice. Our experiments have shown that accurate cost estimates can almost always be obtained with four intervals. Further, the amount of work involved in splitting a vector into a number of intervals equal to a power of 2 is at most double the work involved in splitting it into an arbitrary number of intervals (since an arbitrary number can be rounded up to the next higher power of 2). For a small number of intervals, this additional work is negligible.

The mean partitions the block access distribution more effectively than an approach based on percentiles since, paradoxically, it is less affected by clustered values. For example, suppose we must partition the vector <.3,.3,.2,.1,.05,.05> into two intervals. A percentile method would place the partition point between the elements .2 and .3 since 60% of the distribution lies to the left of this point and 40% lies to the right. Unfortunately, this partition does not work well since it breaks up the .3,.3,.2 cluster. In other words, the percentile method is unable to handle situations where a large portion of a distribution is clustered together. On the other hand, the mean method chooses its partition point between .1 and .2 since the mean of this vector is .17. The cluster .3,.3,.2 pulls the mean up but the cluster .1,.05,.05 pulls it down somewhat and thus the partition point occurs between the two clusters. Thus both clusters are left intact.

```
mean(numint,x)
    /* x = vector to be partitioned into numint intervals */
    /* numint must be a power of 2                        */
    if sizeof(x) ≤ numint
    /*create sizeof(x) singlet vectors each with one element of x*/
    /*and (numint-sizeof(x)) vectors whose only element is 0*/
        return(<x1>,<x2>,...,<xsizeof(x)>,<0>,...,<0>)
    else if numint = 2
        find the largest subscript i such that xi < mean(x)
        return(<x1,...,xi>,<xi+1,...,xm>)
    else /* even(numint) */
        find the largest subscript i such that xi < mean(x)
        return(mean(numint/2,<x1,...,xi>),
                    mean(numint/2,<xi+1,...,xm>))
```

Figure 3.2 -- The mean algorithm for partitioning a vector **x** into a set of subvectors. The set's size must be a power of 2.

3.4. A Specific Example

The mean algorithm described in section 3.3 was extensively tested using combinations of the following four parameters:

1. Blocking Factor (records per block): 5,10,20,50,100
2. Distribution: Uniform,Normal,Exponential,Poisson,Zipf
3. Records Retrieved: 10, 25, 50, 100, .005m, .01m, .05m, .1m, .25m, .5m,m, 2m, 3m, 5m,10m (m denotes the number of blocks containing the relation)
4. Blocks: 500, 1000, 10000

The uniform distribution indicates that the access probabilities were randomly drawn from an interval [a,b] and not that each block had an equal probability of being accessed. In a Zipf distribution, the block access probabilities p_i are defined by the formula $p_i = h/i^z$ where $h = 1/\Sigma(1/i^z)$ and z represents a factor of decay.

Figures 3.3 and 3.4 show representative results for a relation with a blocking factor of 50 records per block and 500 blocks (i.e., a relation with 25,000 tuples). In figure 3.3 we have employed expression (3.1) to plot the expected number of block accesses that occur as the number of records retrieved by a query increases. The query's block access distribution is Zipf distributed. In figure 3.4 we have plotted the percentage differences between the estimates produced in figure 3.3 by expression (3.1) and the estimates produced by the mean algorithm.

The Zipf distribution was used to illustrate the impact that increasingly diffuse block access distributions have on the estimates of the number of blocks retrieved by a type 1 query. Intuitively, the decay parameter *z* can be thought of as an indication of the amount of correlation that exists between the queried attribute and the clustered attribute. As the correlation increases, the *z* parameter increases and the block access distribution becomes increasingly diffuse. The curve plotted in figure 3.3 indicates how the number of block accesses decline as the correlation between two attributes increases.

Figure 3.4 demonstrates that a four interval distribution generates significantly more accurate approximations for expression (3.1) than a two interval distribution. For moderately correlated attributes (e.g., z = 1), the use of only two intervals produced relatively accurate estimates, even when the number of records retrieved was somewhat small. The percentage difference between the estimates produced by expression (3.1) and the mean algorithm quickly peaked around 25% and rapidly fell under 15% where it remained for the rest of the experiment. The higher percentage errors reflected in figure 3.4 for smaller numbers of records retrieved do not translate into large absolute

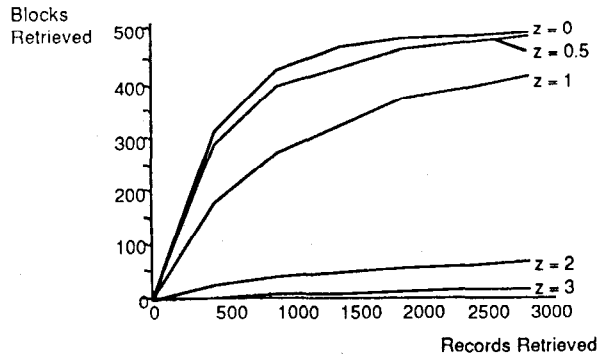


Figure 3.3 -- Expected number of blocks accessed by a type 1 query that retrieves k records. The block access distribution of this query is Zipf distributed with decay factor z.

errors. A typical example might be a block estimate of 25 as compared with an estimate of 20 produced by expression (3.1). This difference would not affect a query optimizer's decision.

As correlation between the clustering and queried attribute increased, the accuracy of the two interval estimate quickly decreased. The error curves corresponding to a decay factor of 2 in figure 3.4 illustrate this deterioration in performance. The error initially peaks around 50% for 100 retrieved records, decreases rapidly to about 15% when 500 records are retrieved, and then steadily rises until it reaches 60% for 3000 records retrieved. Although the estimates produced by only two intervals seem unlikely to mislead the query optimizer, it seems advisable to use four intervals in this instance. The maximum error produced by a four interval estimate is about 15% and the extra flops required to compute such an estimate are negligible.

4. CONSTRUCTING A BLOCK ACCESS DISTRIBUTION

In section 3, it was demonstrated that the mean algorithm accurately estimates the number of accessed blocks when the block access distribution is known. However, in production settings, each query will confront an optimizer with a multitude of different and *unknown* block access distributions. Thus the best an optimizer can hope to do is to maintain a representative block access distribution that can be modified based on the type of query it is presented with. In this section we will show how such a distribution can be constructed and modified using nonparametric techniques.

Two factors have an impact on a query's block access distribution--the size of the query (expressed in terms of the number of tuples retrieved) and the correlation between the queried attributes and the clustering attribute. If the correlation between these attributes is relatively low, then each block has an approximately equal chance of being accessed, regardless of the size of the query. In these situations the independence assumption produces accurate block approximations since it uses a block access distribution in which each block has an equal probability of being accessed. However, as the correlation between the queried and clustering attributes increases, the block access distribution becomes dependent on the size of the query. Small queries tend to have highly diffuse distributions since it is likely that only a few blocks will be accessed. Thus several blocks will be accorded a high access probability while the remaining ones will be accorded negligible access probabilities. As the size of the query increases, the block access distribution tends to become flatter because more blocks are likely to be accessed. Since a larger number of blocks have non-negligible

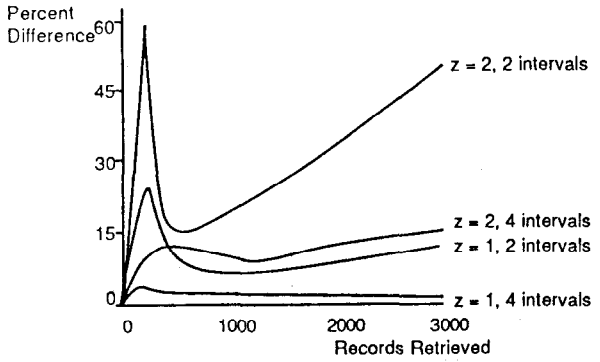


Figure 3.4 -- Percentage difference between the block estimates produced by the majorization technique with two and four intervals and expression (3.1). The block access distribution of this type 1 query is Zipf distributed with decay factor z.

access probabilities, the probability of accessing any individual block decreases, thus flattening out the access distribution.

Ideally, then, the block access distribution should be modified based on the size of the query and the correlation between queried and clustering attributes. The following approach for constructing a query's block access distribution takes both of these factors into account. First, the block access distributions associated with the least frequently, most frequently, and average (i.e., equal to the average duplication rate of the attribute) occurring values are empirically tabulated (suppose that these frequencies are given by the vector $\mathbf{k} = \langle k_1,...,k_a \rangle$ where k_i denotes the number of times the ith attribute value occurs and a denotes the number of values in the attribute domain; let k_1 correspond to the most frequently occurring value and k_a correspond to the least frequently occurring value; let k_m correspond to the average occurring value). Next the mean algorithm is used to partition these distributions into cells. These two steps account for correlation between the clustering and queried attributes. An interpolation formula then reallocates the area enclosed by each of these intervals based on the size of the query the optimizer is currently processing. To use this formula, the query optimizer first determines the interval in which the query's size falls-- $[1,k_a]$, $(k_a,k_m]$, $(k_m,k_1]$, or $(k_1,n]$ (n denotes the number of tuples in the relation). It then constructs an appropriate block access distribution for the query using the distributions associated with the endpoints of this interval (the distribution associated with queries of size 1 is <1,0,0,...,0> and the distribution associated with queries of size n is uniform <1/m,1/m,...,1/m>). The algorithm for constructing the query's block access distribution is formalized in figure 4.1.

As shown by the algorithm, the probability of accessing a block that is assigned to the ith cell of the query's block access distribution is equal to a weighted average of the probability assigned to the cells associated with k_a and k_m. Similarly, the number of blocks allocated to this cell is a weighted average of the number of blocks assigned to the cells associated with k_a and k_m. This same procedure can be used if the query's size falls in one of the other intervals listed above.

5. SIMULATION RESULTS

In order to analyze the performance of the approximations developed in sections 3 and 4 in practical settings, we tested two and four interval estimates using the simulation described in section 2. Recall that the simulation modeled a multivalued dependency A->->B and that for each value in the domain of attribute B, the number of blocks that contained that value was recorded. In addition, the correlation between attributes A and B

without loss of generality, assume $k_a \leq k_q \leq k_m$

let u_{iq} = number of blocks in the ith cell of the query's block access distribution

p_{iq} = probability of accessing a block in the ith cell of the query's block access distribution

k_q = number of tuples retrieved by the query

weight = $(k_m - k_q) / (k_m - k_a)$

for i = 1 **to** numint **do**

$u_{iq} = u_{ia} * \text{weight} + u_{im} * (1 - \text{weight})$

$p_{iq} = p_{ia} * \text{weight} + p_{im} * (1 - \text{weight})$

od

Figure 4.1. Algorithm for constructing the block access distribution of a query that retrieves at least k_a tuples but no more than k_m tuples

was varied from 0 to 1. In this section we will also compare the performance of the two and four interval estimates with the performance of the independence assumption. Since the independence assumption uses a one interval estimate, this section actually evaluates the improvement that results from increasing the number of intervals from one to two or four.

5.1 Results

In figures 5.1 and 5.2 we have reproduced the results for the actual number of block accesses that were displayed in figures 2.1 and 2.2. Figure 5.1 compares the actual number of block accesses required to retrieve 300 records against the number estimated by expression (3.2) when two and four intervals block access distributions are used. The figure assumes that the correlation between the clustering attribute and queried attribute is varied between 0 and 1. When the correlation is greater than 0.5, the four interval estimate performs better than the two interval estimate, providing estimates that vary from the actual number of block accesses by no more than 5%. On the other hand, the two interval estimate provides better estimates when the correlation is less than 0.5. When this condition applies, its estimates vary from the actual number of block accesses by no more than 10%. However, comparing figures 2.1 and 5.1, we find that even better estimates can be obtained if the independence assumption is used when the correlation is less than 0.3.

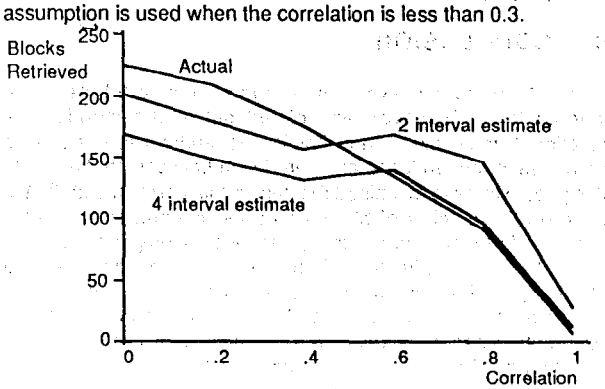


Figure 5.1 -- Comparison between the number of block accesses predicted when attributes are assumed correlated and the actual number of block accesses required to process a query that retrieves 300 records. The correlation between the clustering attribute and the queried attribute varies between 0 and 1.

Figure 5.2 compares the block estimates of the two and four interval estimates when the number of records retrieved is varied between 0 and 1000 and the correlation between the clustering and queried attributes is fixed at .8. The four interval estimate is consistently more accurate, never overestimating the actual number of accessed blocks by more than 10%. By comparing figure 5.2 with figure 2.2, we find that the four interval estimate is also preferable to the estimate produced by the independence assumption.

While figures 5.1 and 5.2 provide specific examples of how the two and four interval estimates compare, figures 5.3 and 5.4 provide a comprehensive analysis of how these estimates compare, both against one another and against the independence assumption (labeled as the one interval estimate in these figures). In figure 5.3 we have plotted the percentage of cases in which each of these three schemes provide the most accurate block estimates. This figure was constructed by analyzing each of the 51 attribute values in B's domain and determining which of the three estimates came closest to predicting the actual number of blocks occupied by each of these values. We can conclude that the independence assumption (one interval estimate) is most appropriate when the correlation is less than 0.4, the two interval estimate is preferable when the correlation is between 0.4 and 0.6, and the four interval estimate is best when the correlation is greater than 0.6. Since the correlation associated with both functional and multivalued dependencies is often greater than 0.4, either the two or four interval estimate will be preferable in many database settings.

Another, perhaps more reliable, way to compare the one, two, and four interval estimates is to compare the actions an optimizer might choose under each of these three schemes. Figure 5.4 provides one such measure--the percentage of cases in which each assumption leads the query optimizer to choose an inappropriate strategy. In this case we assumed that the optimizer could choose between an index lookup and a relation scan to process its queries. The breakeven point between these two options will vary from system to system but for the sake of comparison we arbitrarily set it at 200 blocks. The breakeven point includes only those blocks that contain data which satisfy the query. It does not include additional blocks that must be accessed such as index blocks in an index lookup . Thus if a query optimizer estimates that fewer than 200 blocks contain the required data, it chooses an index scan; otherwise it chooses a relation scan. As figure 5.4 indicates, the independence assumption causes the optimizer to choose the wrong strategy in more than two-thirds of the cases when the correlation is close to 1 and in more than one-third of the cases when the correlation is

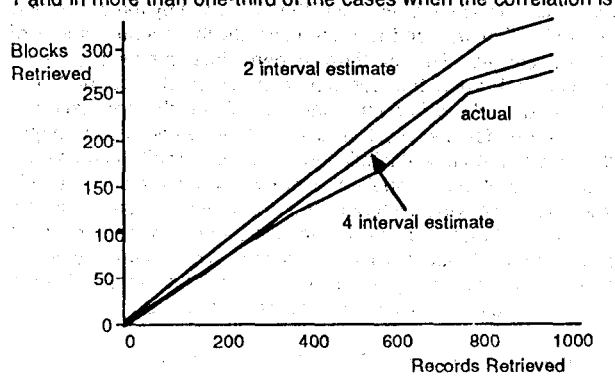


Figure 5.2 -- Comparison between the number of block accesses predicted when attributes are assumed correlated and the actual number of block accesses required to process a query that retrieves a varying number of records. The correlation between the queried attribute and clustering attribute is .8.

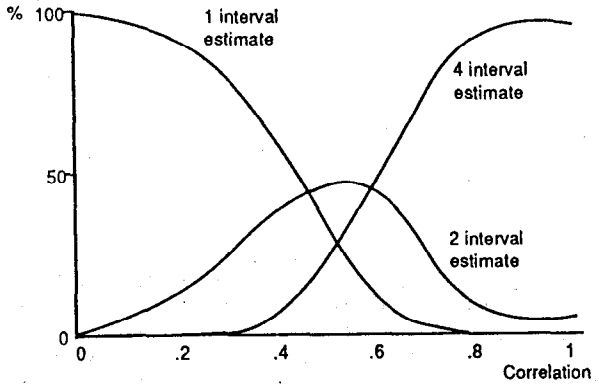


Figure 5.3 -- Percentage of cases in which three different block estimate schemes provide the most accurate block estimates. The one interval estimate corresponds to the independence assumption.

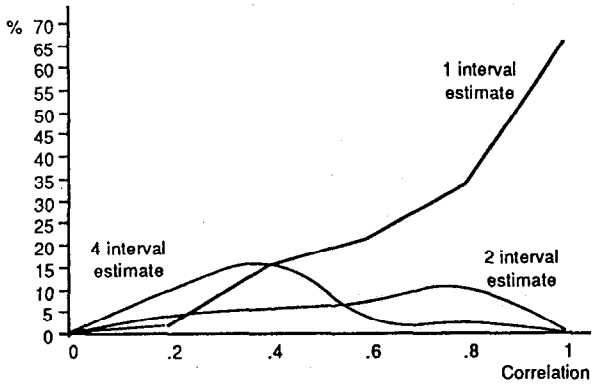


Figure 5.4 -- Percentage of cases in which three block estimate schemes lead a query optimizer to choose a bad processing strategy. The one interval estimate corresponds to the independence assumption.

about .8. This figure drops to 20% for correlations around 0.6 and finally drops to 0 when the correlation decreases to less than 0.2. In contrast the two and four interval estimates mislead the optimizer in at most 15% of the cases and only in restricted regions--the two interval estimate when the correlation is approximately 0.8 and the four interval estimate when the correlation is approximately 0.4. In both cases the number of cases in which the optimizer is misled drops quickly as the correlation moves away from these critical points. Thus by this criteria, the independence approximation should only be used when the correlation is in the range [0,0.2], the two interval estimate should be used when the correlation is between 0.2 and 0.55, and the four interval estimate should be used when the correlation exceeds 0.55.

5.2 Analysis

The failure of the two and four interval approximations to produce accurate block estimates when the correlation between the clustering and the queried attribute is less than 0.4 is attributable to our choice of a block access distribution. Since we assumed that the underlying distribution was not analytically tractible, we tabulated it empirically. However, this decision produces distributions that tend to be more diffuse than the actual distributions. For example, if the actual underlying distribution is uniform (i.e., each block has an equal chance of being accessed), the tabulated distribution will almost surely be diffuse with some blocks having no probability of being accessed and some blocks having a fairly substantial probability of being accessed. Thus the block estimates produced by the correlation approximation will tend to underestimate the actual number of block accesses. Further, these underestimates should be more pronounced when the correlation between the queried and clustering attribute is somewhat low. When the correlation is high, the underlying block access distribution is already highly diffuse so an empirical tabulation does not radically increase the diffuseness of the distribution. However, when the correlation is low, the underlying distribution is fairly uniform and thus the empirical tabulation does increase the diffuseness somewhat significantly. Since our block estimate expressions are Schur concave, we would expect this increased diffuseness to lead to the underestimates that were empirically observed.

Another interesting finding arose when we tried to reconcile the performance of the various estimators in figures 5.3 and 5.4. We were particularly interestested in explaining why the independence approximation seemed feasible when the correlation ranged between 0 and 0.4 in one figure and only seemed feasible when the correlation ranged between 0 and 0.2 in the other. This discrepancy can be explained by examining how badly the independence and correlation approximations can under- or overestimate the number of block accesses required by a query. The estimates produced by the correlation approximations are usually no more than 30% higher or lower than the actual number of blocks retrieved. Further, errors of this magnitude generally occur when well over half the file is being retrieved. In this case, even if the correlation approximation underestimates the number of block accesses by 30%, it often predicts that more than half the file must be retrieved and the optimizer makes the correct decision. In contrast, the independence approximation often overestimates the actual number of block accesses by several hundred percent. In addition, these errors normally occur when less than half of the file is retrieved. But since the independence approximation often overestimates the actual number of block accesses by several hundred percent, the query optimizer often concludes that more than half the file must be retrieved. Thus it often selects a relation scan when it should have selected an index scan. In summary, the independence approximation often produces more accurate block access estimates when the correlation is between 0.2 and 0.4. However, it tends to perform so poorly on small size queries that it causes the query optimizer to choose a poor processing strategy somewhat often. On the other hand, while the correlation approximation tends to underestimate the number of block accesses required by large size queries, the underestimates are not so serious as to cause the query optimizer to choose an inappropriate strategy. Thus the correlation approximation misleads the query optimizer less often when the correlation is between 0.2 and 0.4.

6. CONCLUSION

Most database systems assume that attributes are independent. However, as pointed out in the introduction, functional, multivalued, and implicit dependencies generate a great deal of dependence and correlation between attributes. In this paper we have examined the impact that correlated attributes have on the number of block accesses required to process a query. We found that the independence assumption produces accurate block estimates when the correlation between the clustering attribute and the queried attribute is less than 0.4. As the correlation increases beyond this point, the independence assumption begins to mislead an optimizer or designer in an alarming number of cases, causing them to select inappropriate query execution strategies that could dramatically increase the cost of processing a query. Further even when it produces pessimistic overestimates that do not immediately cause bad decisions, it could distort strategies selected later in a multi-step query operation. Thus when the attribute correlation exceeds

0.4, it becomes necessary to take this correlation into account when generating block estimates.

We proceeded to show that precise analytical estimates could be easily derived when the clustering attribute and queried attribute are correlated but that the expressions which generated these estimates could not be efficiently evaluated. Thus we looked for properties that these expressions satisfied which could be exploited to produce simple approximations. The property of Schur concavity provided us with the necessary insight. Specifically, it allowed us to devise an algorithm that constructs compact block access distributions which can be used to rapidly compute the block cost of a query. We then demonstrated how the original block access distribution could be constructed using several pre-tabulated distributions and the size of the query as input. Finally, we presented the results of several experiments that demonstrated the superiority of our approximations when attributes were highly correlated. Thus the techniques introduced in this paper provide a powerful mechanism for estimating the cost of a query in the presence of correlated attributes.

REFERENCES

Cardenas, A.F. 1975. Analysis and Performance of Inverted Database Structures. *Commun. ACM*, 18, 5 (May), 253-263.

Christodoulakis, S. 1981. Estimating Selectivities in Data Bases. Technical Report CSRG-136, (Dec.), University of Toronto.

Christodoulakis, S. 1983a. Estimating Block Transfers and Join Sizes. In *Proceedings SIGMOD 1983 Conference*, ACM, 40-50.

Christodoulakis, S. 1983b. Estimating Record Selectivities. *Information Systems*, 8, 2, 105-115.

Christodoulakis, S. 1984a. Estimating Block Selectivities. *Information Systems* , 9,1,69-79.

Christodoulakis, S. 1984b. Implications of Certain Assumptions in Database Performance Evaluation, *Trans. Database Syst.* , 9,2 (June), 163-186.

Demolombe, R. 1980. Estimation of the Number of Tuples Satisfying a Query Expressed in Predicate Calculus Language. In *Proceedings of the 6th International Conference on Very Large Databases*, IEEE, New York, pp. 55-63.

Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H.R. Extendible Hashing--A Fast Access Method for Dynamic Files. *Trans. Database Syst* ., 4,3 (Sept.), 315-344.

Feller, W. 1968. *An Introduction to Probability Theory and Its Applications. Vol 1.* 3rd Ed. New York: John Wiley.

Kamel, N. and King, R. 1985. A Model of Data Distribution Based on Texture Analysis. In *Proceedings SIGMOD 1985 Conference*, ACM, 319-325.

Kerschberg, L., Ting, P.L. and Yao, S.B. 1982. Query Optimization in Star Computer Networks. *Trans. Database Syst.*, 7, 4 (Dec), 678-711.

Kotz, S. and Johnson, N.L. 1977. *Urn Models and Their Application*, New York, John Wiley and Sons.

Luk, W.S. 1983. On Estimating Block Accesses in Database Organizations. *Commun. ACM*, 26, 11 (Nov.), 945-947.

Marshall, A., and Olkin, I. 1979. *Inequalities: Theory of Majorization and its Applications.* Academic Press, New York.

Merrett, T.H. and Otoo, E. 1979. Distribution Models of Relations. In *Proceedings of the 5th International Conference on Very Large Data Bases*, IEEE, New York, 418-425.

Montgomery, A.I., D'Souza, D.J., and Lee, S.B. 1984. The Cost of Relational Algebraic Operations in Skewed Data: Estimates and Experiments. In *Information Processing 83* Elsevier North-Holland, New York, 235-241.

Piatetsky-Shapiro, G. and Connell, C. 1984. Accurate estimation of the number of tuples satisfying a condition. In *SIGMOD 84, Proceedings of the Annual Meeting* (Boston, Mass., June18-21). ACM, New York, 256-276.

Siler, K.F. 1976. A stochastic evaluation model for database organizations in data retrieval systems. *Commun. ACM* 19, 2 (Feb.), 84-95.

Vander Zanden, B.T., Taylor, H.M., and Bitton, D. 1985. A general framework for computing block accesses. *Technical Report 85-718*, Cornell University.

Yao, S.B. 1977. Approximating block accesses in database organizations. *Commun. ACM* 20, 4 (April), 260-261.

Zahorjan, J., Bell, B.J., and Sevcik, K.C. 1983. Estimating Block Transfers when Record Access Probabilities are Non-Uniform. *Information Processing Letters* , 16, 5 (June), 249-252.