EFFICIENT MULTI-ATTRIBUTE RETRIEVAL OVER VERY LARGE GEOGRAPHICAL DATA FILES

John L. Pfaltz
Dept. of Applied Math. and Computer Science
University of Virginia
Charlottesville, Va. 22903

Retrieval, or data access, in useful geographical data bases is typically characterized by three conditions:

a. the files are large; we expect over a million records, or items;
b. the files need not be homogeneous; different geographical items may be represented by records of differing lengths and formats; and
c. a desired item, or set of items, may be identified by more than one characterizing attribute.

Data access must take place in this kind of environment.

An attribute of an item, or record, is a descriptive value which may serve to characterize, or identify, the item. Unique identifiers, such as social security numbers, are attributes; but so are descriptive terms such as "city", "road segment", "land parcel", etc. All geographic data files are inherently <u>multi-attribute</u> files, since each item is characterized by at <u>least</u> two coordinates specifying its spatial location. And normally there are many more attributes, such as "land use", "assessed value", "traffic density", "population", etc.; where not all attributes need be specified for a given type of item; where not all attributes need be specified in a given retrieval request, or query. For example, a procedure may issue a query for all items such that:

$36.5 \leq \text{lat} \leq 38.3$, $75.2 \leq \text{long} \leq 79.0$, type = church, and assessed value > 85,000.

Consideration of such typical queries provides additional characteristics desired in an effective geographical data retrieval system.

d. Partial-match retrieval, in which only some subset of the identifying attributes are specified, should be possible.
e. It should be possible to retrieve over a range of attribute values.

And finally, it would be desirable if:

f. as more identifying attributes are specified, the system responds with less computational cost;
g. the system could be "tuned" to respond more efficiently to commonly expected queries, while still retaining the capacity of handling less frequent queries; and
h. the system could handle (essentially) simultaneous queries from the same, or different, users.

Retrieval using <u>indexed-descriptor files</u> appears to provide many of these desirable characteristics.

Any attribute, A_j, may take on a range of attribute values, v_j, appropriate for that attribute. For example, 36.5 and 38.3 may be attribute values for the attribute "lat", as may any real number in the interval (-90.0,+90.0). "Church" may be a value of the attribute "type". We shall use A_j to denote the j-th attribute in the system; and v_j to denote any particular value of that attribute.

A descriptor, D, is simply a bit string (of zeros and ones), conceptually subdivided into f <u>fields</u>. Each field, F_j $(1 \leq j \leq f)$ corresponds to a single attribute defined withing the system. For example, a single descriptor in a system handling only four attributes might look like:

01000 100 000010000 0000001

The width (number of bit positions) in each field is denoted by w_j; the entire descriptor width denoted by w. Readily $\sum w_j = w$.

We will assume the existence of f separate transformations $T_j:\{v_j\} \rightarrow [1,w_j]$, each of which maps a valid attribute value, v_j, in its domain into a single bit of the corresponding descriptor field. These transformations, which should map values in a relatively uniform manner over the field, form the only link between the indexed-descriptor files and the actual attribute values. They are tailored to the particular set of attribute values, and are usually quite easy to construct.

As each item, or data record, R is entered into the system, each of its characterizing attributes $(v_1,v_2,\ldots,v_f)$ are transformed, and bits $T_1(v_1)$, $T_2(v_2),\ldots,T_f(v_f)$ are set to form its associated record descriptor D_R, similar to the one shown above. Each record descriptor will have precisely one 1-bit set in each of its f fields (assuming all attributes are common to all items, fewer otherwise).

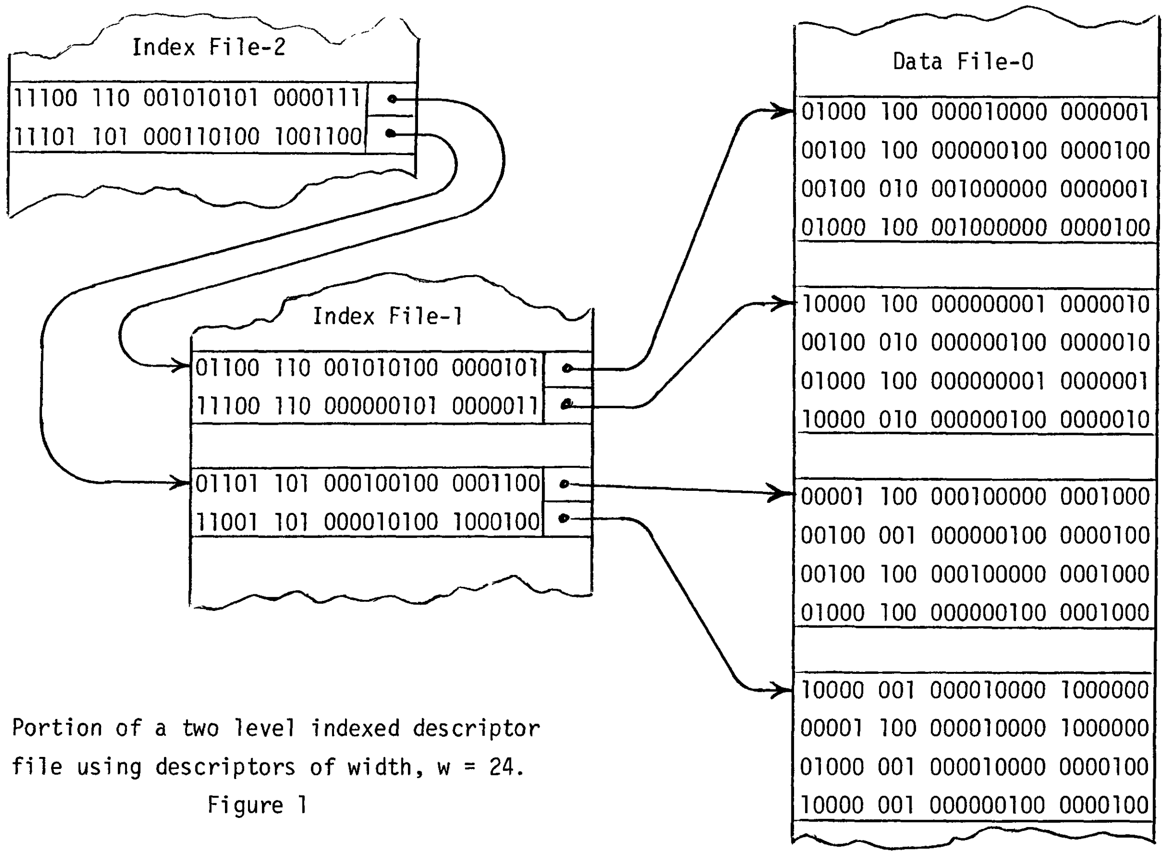
Now consider a typical partial-match query that does not involve range searching. Normally only some subset, say q values, of the possible f attributes are specified in the query. Suppose they are $(v_1', v_2', \ldots, v_q')$. (Here we are being mathematically imprecise, since they need not be the "first" q attributes, they can be any q; but we want to avoid double subscripts.) We will form the associated query descriptor, Q, by transforming each of the specified query values, v_j', setting bit $T_j(v_j')$ in the corresponding field of Q, and resetting the unspecified attribute fields to all zeros (a "don't care" condition). Thus each query descriptor Q will have precisely $q \leq f$ bits set.

Now, using just record descriptors and query descriptors we could design a rudimentary retrieval system. We could create a search file consisting of just the individual record descriptors, together with pointers to the actual storage location of the associated data record. Now we could sequentially compare the query descriptor Q with each record descriptor d in this search file. If $Q \subseteq D_R$ then R may satisfy the query; it must still be accessed and its actual attribute values $(v_1, \ldots, v_f)_R$ compared with the specified query values $(v_1', \ldots, v_q')_Q$. But if $Q \nsubseteq D_R$ then R can not possibly satisfy the query; we may skip on to the next record. This rudimentary retrieval scheme has a few attractive features. First, descriptor comparison is very much faster than the comparison of individual attributes. Also, the fixed length descriptor records (with their pointers) of the search file are very much shorter than their associated data records, and thus can be accessed from storage in larger units. Even so, the sequential scanning of the million, or so, record descriptors in the search file is clearly impractical. But it does suggest an idea which is the conceptual key to forming indexed-descriptor files.

In the data file, which we denote file-0, several data records are packed into a single block, β, and a block descriptor D_β is formed by OR-ing together each of the individual record descriptors, that is, $D_\beta = \bigvee D_R$ for all R in β. Now these block descriptors D_β (together with pointers denoting the storage address of the block) are accumulated to form a sequential search file, which we call file-1. If $Q \subseteq D_\beta$ then the entire block will be accessed and the actual attribute values of each record will be compared with the query values to see if it satisfies the query. But if $Q \not\subseteq D_\beta$ then no record in the block can possibly satisfy the query; all may be safely skipped, and the next block descriptor examined. If r_0 denotes the total number of records in the data file-0, and p_0 denotes the packing (or blocking) factor of records per block, then clearly there will be only r_0/p_0 block descriptors in the index (or search) file-1. "r_0/p_0" may still be too many descriptors to reasonably search in an exhaustive manner; so we may employ the same technique once again. p_1 individual descriptor records in file-1 may be packed into a common block, call it α. Its block descriptor is formed by OR-ing together all of its individual descriptors, that is $D_\alpha = \bigvee D_\beta$ for all D_β in α, and this descriptor together with a pointer to the block α is added as a record to a higher level index file-2. If in searching index file-2, we find that $Q \subseteq D_\alpha$ then the block α must be accessed and Q compared with each D_β in α. But if $Q \not\subseteq D_\alpha$ then none of the $p_0 \cdot p_1$ records subsumed by D_α can possibly satisfy the query and all may be safely ignored. Now file-2 will consist of only $r_0/(p_0 \cdot p_1)$ descriptors. If this is too many for exhaustive search, then it too can be blocked to form index file-3, and so on. Normally the highest level file-d, which must be exhaustively searched for every query, should be small enough to be core resident.

Figure 1 illustrates a small portion of a typical two level indexed-descriptor file formed in this manner---except that the data file-0 would actually consist of much longer data records each consisting of their actual attribute values $(v_1, \ldots, v_f)$ and associated data, not a record descriptor as we have shown; and each block would be very much larger than illustrated.

The structure of indexed-descriptor files, shown in figure 1, arose in response to a retrieval that has been informally stated in the preceding paragraph. We can formalize the query, or search, procedure by means of the following two procedures.



Portion of a two level indexed descriptor file using descriptors of width, w = 24.

Figure 1

```
procedure query ($v'_1,\ldots,v'_q$)
    |Given the set of q specified attribute values,|
    |apply the transformations to form a query     |
    |descriptor with q non-zero fields.            |
    $Q \leftarrow \{T_j(v'_j)$ : for specified query attributes, j$\}$;
    |Exhaustively search all descriptors in the |
    |highest level index file-d.                |
    for each $D_\beta(d)$ in file(d) do
        if $Q \subseteq D_\beta(d)$ then search($\beta$,d-1);
    end
recursive procedure search (beta,level)
    fetch block beta from storage;
    |Examine all records or descriptors in this block.|
    if level > 0
       then |This is a block of descriptors in |
            |an index file.                    |
            for each $D_\beta$(level) in block beta do
                if $Q \subseteq D_\beta$(level) then search($\beta$,level-1)
       else |This is a block of data records.|
            for each R in block beta do
                if $(v_1,\ldots,v_f)$ satisfies $(v'_1,\ldots,v'_q)$
                   then add R to response set
    end
```

How efficient is this retrieval procedure? Each call to the procedure "search" involves the random access of a block in one of the indexfiles 1,...,d-1, or the data file-0. And the cost of such access into secondary storage far outweighs the relatively minor cost of sequentially examining the block in core. So we will measure retrieval efficiency solely in terms of the expected number of storage accesses. Let Q be any arbitrary query. The probability that, for a given block descriptor D_β, $Q \subseteq D_\beta$, (thereby requiring access to the block β) is simply the product of the probabilities that in each of the q specified fields of Q, the single bit $T_j(v'_j)$ matches a bit set in field-j of the descriptor D_β. Let $\overline{b}_j(i)$ denote the average (or expected) number of bits set in field-j of all descriptors in file-i. Then $\overline{b}_j(i)/w_j$ will denote the probability that field-j of Q is contained in field-j of D_β. And $\prod_{j\in Q} \bar{b}_j(i)/w_j$ denotes the probability that $Q \subseteq D_\beta$, where D_β is an arbitrary block descriptor in file-i.

Let r(i) denote the total number of records in file-i. With the basic probabilities given above, it is not difficult to perform a straightforward derivation (involving simple conditional probabilities) that yields the following expression for the expected number of storage accesses, given an arbitrary query Q:

$$E(\text{blocks accessed}|Q) = \sum_{i=0}^{d-1} r(i+1)\left[\prod_{j\in Q} \bar{b}_j(i+1)/w_j\right]. \qquad (1)$$

Examination of expression (1) reveals that retrieval costs are minimized when there are relatively few records in the index files (that is, when blocking factors are relatively large); when the density of bits in descriptor fields, $\bar{b}_j(i+1)/w_j$, is small (that is, fields are wide and few bits are set by the OR-ing process in blocking); and when several attributes are specified in the query (so that the multiplicative factor on the right decreases exponentially). Unfortunately, these desirable characteristics are frequently incompatible, so that crucial tradeoffs must be chosen in the course of the file design.

Perhaps the best feeling for the significence of expression(1) can be gained from a real example. The only large data base currently employing indexed file retrieval has been built by Ed Cagley, and his associates, at the Mathematics and Computation Laboratory of the Federal Preparedness Agency---to whom credit must be given for conceiving the method. They have a dynamic file of approximately r(0) = 1,440,000 records. These are packed with 24 records per block, yielding r(1) = 60,000 descriptor records in index file-1. These inturn are packed with 128 descriptors per block, so that there are r(2) = 470 descriptor records in file-2. Since this is reasonably core resident, the entire file has depth, d = 2.

Records in this file are characterized by seven attributes, and most queries specify all seven. Using average observed bit densities in fields 1 through 7 of the two index files of this system, we find that $\prod_{j=1}^{7} \bar{b}_j(2)/w_j = .00249$ and that $\prod_{j=1}^{7} \bar{b}_j(1)/w_j = .0000544$. These values serve as the right hand factor in expression (1). Substituting in all these values, we get

$$E(\text{blocks accessed, or disk accesses}|Q) = 4.436$$

which accords closely with observed behavior. Normally 3 or 4 disk accesses are sufficient to retrieve a single, fully specified, record.

Frequently, and especially in geographical information retrieval, not all of the attributes of a record are specified in the query. When only a subset of the possible attributes are specified, yielding a partial match retrieval, one expects, possibly many, records of the data file to satisfy the query criteria. If the relative frequency of specific attributes appearing in queries can be anticipated, then the system can be "tuned" to respond optimally with respect to those attributes---at the cost of less optimal response to less frequently used attributes. This "tuning" can be performed by varying the widths, w_j, of descriptor fields, or by carefully organizing the records within blocks of the data file. In Cagley's file, if only 3 of the 7 attributes are specified, one can expect that from 1200 to 1500 records will satisfy the query (regardless of which three are specified). Given the optimization algorithm he employs, if the three most important attributes are specified then an expected 61.8 disk accesses will retrieve <u>all</u> the records. If the three least important attributes are specified then an expected 1966.2 disk accesses will be required to retrieve approximately the same number of records. This represents a wide variance in expected behavior (nearly 30-fold); but this example illustrates both the benefits that can be obtained by tuning, and the fact that at its worst, indexed descriptor retrieval is still quite acceptable.

At the onset, we emphasized the importance of range search queries in geographical systems. Then for reasons of presentation we ignored it. To implement range search capability we first require that the transformations T_j, which map attribute values into descriptor bit positions, be <u>order preserving</u>. That is, we require $T_j(v) \leqslant T_j(v')$ only if $v \leqslant v'$. (Reasonable order preserving transformations are possible because the efficiency of the retrieval process is not predicated on a uniform distribution of transformed values, as, for example, in hash-code retrieval techniques.) Now if a query specifies an attribute of the form $v_j \leqslant A_j \leqslant v'_j$, then the bits $T_j(v_j)$ and $T_j(v'_j)$, and all bits inbetween, will be set in the query descriptor Q. Since more than one bit can be set in any field of Q, we can no longer use the simple test, "is $Q \subseteq D$?" in the search procedure. Nor can we set "don't care" fields of Q to all zeros. Instead, "don't care" fields of Q are set to all ones; and the search test becomes "is $\text{field}_j(Q) \wedge \text{field}_j(D) \neq 0$, for all $j \in Q$?". With this it is not hard to work out the details of a retrieval procedure embodying range search capabilities.

As yet, we have been unable to combinatorially analyze the expected behavior of range search queries. However, empirical observations on small test files of 5,000 to 10,000 records indicate the practicality of this retrieval mode.