A Review of Recent Work on Multi-attribute Access Methods

David Lomet
Digital Equipment Corporation
Cambridge Research Lab
One Kendall Square, Bldg. 700, Cambridge, MA 02139

Abstract

Most database systems provide database designers with single attribute indexing capability via some form of B+tree. Multi-attribute search structures are rare, and are mostly found in systems specialized to some more narrow application area, e.g. geographic databases. The reason is that no multi-attribute search structure has been demonstrated, with high confidence, to be effective for a broad range of applications. Multi-attribute search is an active area of research. This paper reviews the state of this field and some of the difficult problems, and reviews some recent notable papers.

1 Access Methods in Database Systems

Indexing provides an efficient way to selectively access some records within a large collection of records associatively. When using an ordering access method, a distance metric on values for the keys(attributes) is preserved. This permits an ordering access method to efficiently support range searches for records with keys in some key interval. For example, one might ask for all people in a directory with names between "Smith" and "Smithson". An ordering access method would perform this range search in time linear in the number of names in the range, after an initial search logarithmic in the size of the directory.

Ordering access methods applicable to arbitrary sets of key values are based on tree search. The B-tree [1, 2] is a very robust ordering access method that is ubiquitous in database systems. It is a paginated search tree with high fanout nodes that is used for one dimensional search spaces. B-trees grow by the splitting of overfull nodes, followed by the posting of index terms higher in the tree. Searches touch only $log_{fanout}(filesize)$ pages (approximately), which is a factor of eight improvement over binary search when fanout is 256, a typical value. Storage utilization with node splitting is about 69%.

Over the past twenty years, B-trees have been greatly refined. Things like main memory buffering, key compression, partial expansions, and large bucket size have all been exploited. A more complete review is given in [6]. Our assessment is that there is little more to gain in performance attributes for one dimensional access methods. The areas that remain of further research interest are in other areas, e.g. (i) concurrency control and recovery with transaction semantics; and (ii) on-line and high speed (parallel) construction of B-tree indexes.

2 Multi-attribute Indexing

Multi-attribute searching, either for point data or spatial data, adds a new set of problems that did not arise in single attribute methods. These problems, and possible approaches are discussed below.

2.1 Multi-attribute Point Indexing

2.1.1 Dividing Multi-attribute Space

Partitioning a multi-dimensional space containing point data is usually possible by choosing a convenient attribute and value as a boundary, much as is done for one dimensional B-trees. That usually produces relatively even allocation of points to the resulting regions. Some special cases exist in

which data consistently falls on potential boundaries, which can sometimes make this harder. But the real difficulty for these indexing methods is the problem of how to split non-leaf (index) nodes of the tree.

While data of a search space may be points, index terms always reference subspaces of the search space. Regions of higher dimensional spaces can have more complex boundaries than one-dimensional regions. Complex boundaries of the indexed subspaces make it difficult to find a simple boundary that cleanly separates the index terms into two disjoint regions.

The k-d B-tree [13], an early multi-attribute paginated search structures, uses only a single attribute value as the boundary. One can always make this work. This is very simple and search space partitioning is very effective for range searching. There are some dangers, however. Utilization will suffer if the splitting is very uneven. Further, downward cascading splits are required if the split boundary crosses the subspace of an index term, which is common.

2.1.2 Mapping to Single Attribute Space

A second approach to multi-attribute indexing is to map multi-attributes spaces to single attribute spaces by defining a one dimensional "walk" through multi-attribute space. Z-order [9] does this by interleaving the bits that represent the attribute values together into a single bit string. A single dimensional index method, e.g. B+trees, can then be used to search the space. This is simple and robust. Also important, it permits an existing one-dimensional access method to be used. A difficulty is that the natural specification of multi-attribute ranges must be mapped to a collection of ranges that approximate the naturally specified range. Figure 1 shows the "walk" through a two dimensional space that results from the z-order transformation. Note how compact regions of the space can become non-contiguous regions of the z-order linearization.

2.2 Spatial Indexing

Spatial data has extents (areas or volumes). Hence, unlike point data, spatial data can overlap. Overlapping makes it impossible to successfully partition a multi-attribute search space containing spatial data by traditional node splitting. The result is that spatial indexing is more complex than multi-attribute point indexing.

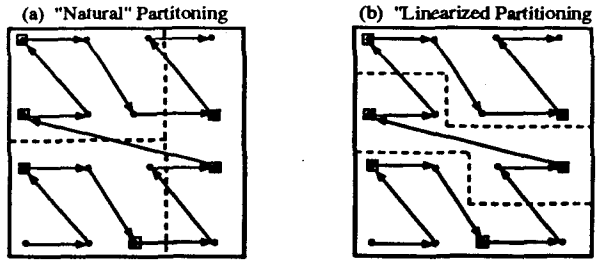

Figure 1: The points in 2-space are arranged in a Z-order walk. The squares represent data points. A "natural" partitioning of the original space is shown here to be different from the partitioning achieved when working on the Z-ordered space.

Two ways of treating spatial data are discussed below. Spatial data can be indexed "naturally" or it can be mapped to "parameter" space, where its boundary values are treated as a higher dimensional point.

2.2.1 Native Space Indexing

Native space indexing preserves the distance metric of the original space. That is, objects close in the underlying native space will be close in the space as indexed. A spatial object is mapped to every subspace that overlaps with its extent.

Due to spatial data overlap, a clean split, which partitions the space of a node and which results in data going to exactly one of the regions of the partition, may not be possible. The earliest method to deal with this was the R-tree [3]. Rather than partitioning the search space, R-tree index terms in a node can refer to overlapping subspaces. Search for a single point in an overlapped region will need to do multiple tree traversals, a substantial negative.

Node splitting in the R-tree is complex and hard to optimize. Indeed, there have been several papers on R-tree node splitting. As the number of regions indexed grows, there tends to be increasing overlap among the regions, leading to increasing overlaps among the subspaces indexed. This can

lead to reduced performance, sometimes dramatically reduced.

The spatial overlap problem can also be solved by partitioning the space and storing a copy of the data in each subspace of the partition that it intersects. This is called "clipping". It is the approach used for R+trees [14]. Clipping avoids multiple tree traversals. Range searching is simple as a single node contains all data relevant to a subspace. However, a mix of large regions with small can lead to substantial duplication of the large regions. Further, if all data of a node intersect the entire space of the node, then node splitting is no longer effective and another technique needs to be exploited. This difficulty increases as the number of regions indexed increases.

2.2.2 Parameter Space Indexing

One can map spatial data to point data by circumscribing a spatial object with a simple containing region, e.g., a bounding box. Then, high and low values for each box coordinate can be used as indexing attributes. There are twice as many bounding values as there are dimensions to the original space and so the dimension of this "parameter" space is twice that of the native space. A multi-attribute point access method may then be used, avoiding the problems of overlapping areas and complicated boundaries. Because objects are mapped to points, they can be uniquely assigned to exactly one of the regions of a partitioned parameter space. Also, non-spatial attributes can be indexed together with the spatial ones within a single tree, becoming simply additional attributes of the multi-attribute point. For example, one might distinguish photos of a region by time. (Native space indexes are usually described as purely spatial indexes.)

Parameter space does not preserve the metric of the native space. Objects close in native space can be at some distance in parameter space. However, all answers to the question of which regions of the search space contain spatial data that overlap a given subspace are in a compact region of the mapped search space, which can be found using a range search. See Figure 2 taken from [6], which also has a more complete discussion of the advantages of parameter space. Parameter space

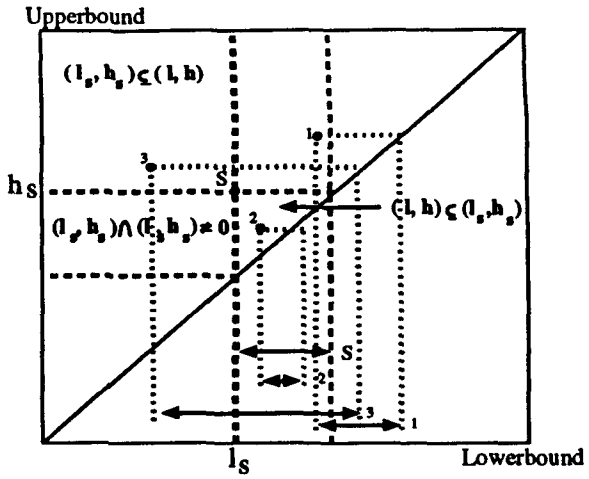

Figure 2: The mapping of linear subspaces to points effectively clusters the answer sets for containing, contained in, and intersection queries. The answer regions for these queries are rectangular, and hence the borders of the answer sets are short.

indexing achieves this compactness while avoiding the problems of overlapping spatial objects.

Bounding boxes usually only approximate real spatial data. Thus, the index is acting as a filter, delivering candidates for further inspection. Note, however, that this "filtering" is common to access methods. For example, a B+tree returns a page on which a requested point resides. The point data on the page must then be refined, i.e. examined in detail, to find the desired data. There are no missed answers, only false drops that are eliminated by subsequent refinement.

It is tempting to think of applying the parameter space mapping recursively to the subspaces represented by index terms. However, this introduces the need for a range search at each step of a tree traversal for search. That the subspaces of index terms already are disjoint (except for R-trees), removes some of the benefit of the mapping. Thus, one does not want to map the area described by an index term to a higher dimensional point.

2.3 Multi-attribute Joins

2.3.1 Point Joins

Single attribute indexes can be very effectively exploited for join processing. This is especially true

for equality joins. An index on a table permits one to designate that table as the "inner" table of the join, and to perform a highly optimized nested-loops join, in which an index search replaces the inner loop.

How to exploit a multi-attribute index in order to perform point joins, and equi-joins specifically, is little studied, no doubt because of the near absence of multi-attribute indexes. If all attributes of the index are specified in the join, then the index can be used analogously to the index for the single attribute case. But subsetting the attributes results in a partial match query and introduces the need for range searches, which increases the cost of the join. Analysis of query costs may then show an advantage for other join algorithms.

2.3.2 Spatial Joins

Spatial joins are quite different from the multi-attribute equi-joins discussed above. An example might be to join a cities table with a rivers table to find the cities that are on rivers. The study of spatial joins remains in its infancy.

A spatial join can be defined as follows. Given two sets of spatial objects, R and S, the spatial join of R and S is the set of pairs, [r, s], such that r is in R, s is in S, and r and s overlap spatially. If R is a single region and S contains data points, then spatial join reduces to a range query.

Join processing can be done in native space or parameter space. Further, one can exploit various linearizations to reduce the processing to one dimension. Interestingly, when that is done, it becomes possible to use a form of sort-merge join (see below).

3 Recent Work

In this section, we briefly review recent papers that explore the indexing of multi-attribute point and spatial data. The papers build on the themes that have been discussed above. The papers are grouped by topic area, and within these groupings, the paper title is used as a subheading.

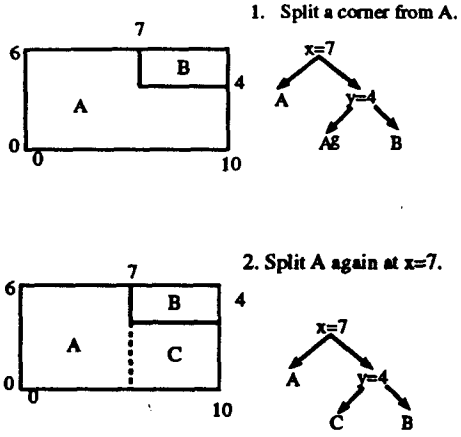

Figure 3: The k-d tree is used to keep track of how holey bricks have been split.

3.1 Multi-attribute Point Indexing

3.1.1 Multi-attribute Boundaries

The hB-tree [8] is derived from the k-d B-tree. Its goal is to avoid downward cascading of splits, hence avoiding both re-structuring cost and adverse storage utilization. The technique it uses is to remember, via a k-d tree stored within an index node, the space decompositions that were performed in forming its child nodes. It then chooses one or more of these existing boundaries as the boundary for splitting the index node when it becomes full. This results in the removal of a smaller brick-like subspace from the larger brick-like subspace of the original full node. The result is what is called a "holey brick". Figure 3 shows how a k-d tree is used to represent these holey bricks.

While the hB-tree has many similarities to B+trees on a global level, node splitting is different. Splitting extracts a region described by a subtree of the k-d tree. Highly skewed distributions are dealt with in two ways:

1. good storage utilization is ensured by permitting any k-d tree node to be the root of the extracted tree, not only subtrees that correspond to spaces defined by child index terms.

2. an index term (which is a k-d tree fragment) for a node is kept small by including in it only the k-d tree nodes whose attribute values specify boundaries for the node's region, regardless

of how far down in the k-d tree the extracted region is. Size of an index term is thereby limited to 2k+1 k-d tree nodes.

Both of the above techniques complicate splitting and the resulting space decomposition. Indexed areas are clipped, and the space decomposition at one level of the tree differs from the decomposition at other levels. An hB-tree node will occasionally have multiple parents.

The complex splitting has the positive result that the hB-tree has robustness properties similar to B+trees, namely good worst case guarantees for index node fanout and storage utilization. The k-d tree used to remember the space decompositions of child nodes also becomes the intra-node search structure, speeding search. This k-d tree is a very compact representation of the subspaces of child nodes, enhancing index node fanout.

3.1.2 Improving Linearization of Multiple Attributes

Any linear "walk" through a multi-dimensional space will result in some points close together in the higher dimensional space being at some separation. The original z-order has a second attribute, namely that at times, points adjacent in the linear space will be at some distance in the higher dimensional space (see Figure 1). This later characteristic is not an essential one.

In [5], a linearization of 2-space was described that uses a "Hilbert curve". The important aspect of the Hilbert curve is that its adjacent points are all also adjacent in the higher dimensional space. This has potentially important implications for the specification of search ranges and the performance of range searching. A disadvantage is that the complexity of transforming a point in 2-space to its Hilbert curve representation is substantially greater than the cost of the bit interleaving required of z-order.

A series of simulations were reported in [5], in which search subspace size and shape were varied. The results showed the Hilbert curve to have a consistent though modest advantage for range search over z-order and a number of other linearizations. In particular, the number of disk blocks fetched in range search and the number of times that the fetch was to non-adjacent disk blocks were both reduced. Particularly important, the simulations showed that the number of disk blocks fetched was a function of the amount of data in the subspace queried, not the size of the database. This applies to z-order as well.

3.2 Spatial Joins

3.2.1 Using Z-order

[10] contains a z-order based algorithm for spatial join. Each spatial object is decomposed into rectangular regions through a recursive splitting of the space containing the object. The orientation and position of the splits are such that each region resulting from the decomposition is represented by a range of z-values. The decomposition yields one or more ranges of z-values, z1, ..., zn. The spatial object defined by the union of the regions corresponding to z1, ..., zn, contains the object. That is, the z-order representation is a conservative approximation of the object. Each spatial object o is represented by the pairs [z1, o], ..., [zn, o].

This spatial join algorithm is essentially a sort-merge join of two z-ordered sequences, R and S. The [z, o] pairs for all spatial objects in each input are sorted by z-value. A two step process is applied to each pair.

1. A filter step involving only z-value comparisons locates candidate pairs [r, s] for which a z-value of r contains, or is contained by, a z-value of s. This indicates that the spatial objects are likely to overlap.

2. A refinement step checks each candidate pair for actual overlap using the precise dimensions of the objects, not the z-values.

Performance is determined by the time required to carry out the filter step, (usually measured in terms of pages accesses), and the accuracy of the filter step. Accuracy is important because it determines the time required by the refinement step.

An input to a spatial join may often be localized in a few areas, e.g. the query rectangle in a range query. These clusters are usually preserved in the linear version of the search problem. If a

z-value index exists for either or both of the sequences, a range search on z-value can eliminate non-productive sequential accesses to regions that have no overlap candidates. This is analogous to an index join for single attribute joins.

3.2.2 Redundancy of Representation

In deriving the z-order representation of an object, one can choose how many ranges of z-values to use. The more ranges, the closer the approximation. In essence, the z-order grid is being used to "clip" the object being represented. Redundancy of the z-order representation of a set of objects is defined as the number of z-values resulting from the decompositions of the objects divided by the number of objects.

Redundancy is greater than or equal to one (one z-value per object) and its upper bound is determined by the resolution of the grid on which the space-filling curve is defined. In [11], the speed and accuracy of the filter step is shown to depend on redundancy. Low redundancy approximates the spatial objects poorly. The effect is to make z-valued index access inefficient, the filter step slower than it could be, and accuracy of the filter step low.

As redundancy is increased, there is rapid improvement in both speed and accuracy of the filter step. Eventually, filter step performance reaches a maximum. Beyond this point, accuracy continues to improve (although more slowly), but the cost of the filter step increases, as the high redundancy causes retrieval of many copies of the same objects. One can control this redundancy with a parameter.

3.2.3 Native vs Parameter Space

Z-order based join algorithms can be applied to any kind of spatial or point object. Hence, they can be applied to parameter space as well as to native space representations. [12] compares these approaches for spatial joins. The performance results are not clear-cut, but depend on the value of $min(|R|, |S|)$, and the size of the objects.

Because of the reliance on filtering, z-order join algorithms have poor worst-case performance. Nonetheless, experimental performance results have been good. However, they have been based, for the most part, on artificial data.

3.3 Performance Studies

3.3.1 Experimental Study

Evaluating spatial access methods presents a multiplicity of problems that are not present when dealing with one dimensional indexing. In addition to how data should be distributed, the size and shape of the regions that are represented by the data need to be supplied. There are no widely accepted analytic workloads for spatial structures. As a result, performing experiments on data that is "real" for some application has a credibility that simulations with synthetic data frequently lack. A recent SIGMOD Conference paper, [4], is illustrative of papers performing comparative studies on real data.

In [4], the source of the data is the Bureau of Census TIGER/Line file for representing the roads and geographic features of the United States. The subset of the Census data used covered counties in Maryland, and was primarily line segment (roads) in form. The experiments compare the R-tree, the R+tree, and the PMR-tree, the latter being a tree structure derived from the quad-tree, but in other respects reminiscent of the R+tree in that clipping is used on the native space representation of spatial data.

The experimental results were quite carefully qualified in their presentation, which should be a model for other more enthusiastic researchers who sometimes overstate their results. The results demonstrated some search performance advantage for the PMR tree, with some penalty in space required for this tree.

3.3.2 Analysis and Simulation

Obtaining good analytic results for multi-attribute or spatial indexing has been very difficult. One example of such an analysis, with a confirming simulation study was done on the time-split B-tree or TSB-tree [7]. The TSB-tree is a tree that maintains an index on versions of data by both key and transaction time. A TSB-tree node can be key-split, essentially like B-trees, or time-split with clipping (duplicating outdated versions). A special property is that only current versions of data, not outdated versions, can be updated.

The non-updatability in the time dimension sim-

plifies both the TSB-tree and the resulting analytic study. All versions of data in the index start out as current versions. By varying the ratio of insertions to updates, one can control the ratio of outdated to current versions. In [7], fringe analysis, the analytic method of choice for one dimensional access methods, was used. Several forms of storage utilization results were derived, along with quantities that contributed to these results. These were graphed as a function of the ratio of inserts to updates.

A simulation in which a synthetic workload was applied to a skeletal TSB-tree implementation confirmed the analysis. It extended the numeric results to larger node sizes as well. The paper ended with a presentation of conclusions that lent some insight into why the TSB-tree performed as it did.

This style of performance paper follows the model for one dimensional access methods and represents a beachhead in the harder area of multi-attribute methods. Additional work will be required before analytic methods deal well with more general multi-attribute data.

4 Discussion

No one method of multi-attribute and spatial indexing has become a standard in the way that B-trees did for single attribute, tree-based access methods. Node splitting is simply harder in higher dimensions. This has resulted in a multiplicity of splitting techniques that make different trade-offs between robustness, simplicity, and hoped for performance.

The increased degrees of freedom in multi-attribute space have made analysis harder. Applying fringe analysis to a multi-attribute access method has not yet been done. An analytic description of an acceptable workload is required for this. Also lacking is a widely accepted experimental workload. This problem may be resolved via the creation of the Sequoia 2000 benchmark [15], which will include experimental workloads based on real data and will be publicly distributed.

Only limited work has been done on using multi-attribute indexing for joins. Exploiting indexes for spatial joins is critical to the performance of geographic applications. Research has only begun in this area.

One needs to make multi-attribute access methods highly concurrent and recoverable if they are to become parts of general purpose database systems. This problem has been addressed for some classes of trees, but most multi-attribute access method papers have not dealt with the issue.

How or whether to transform spatial data for subsequent indexing or joining remains an open issue. Should parameter space be used for spatial data, eliminating the overlapping region problem? Should a linearizing transformation, e.g. z-order, be used to eliminate entirely the need for multi-attribute access methods by performing multi-attribute indexing with a single attribute access method? Definitive answers to these questions do not yet exist.

5 Acknowledgements

Jack Orenstein and Betty Salzberg have been instrumental in the writing of this paper. They spent time educating me and provided specific comments that helped improve the presentation.

References

[1] Bayer, R. and McCreight, E. Organization and maintenance of large ordered indices. Acta Inf. 1,3(1972), 173-189.

[2] Comer, D. The ubiquitous B-tree. ACM Computing Surveys 11,2 (June 1979), 121-138.

[3] Guttman, A. R-trees: a dynamic index structure for spatial searching, Proc. ACM SIGMOD Conf. (Boston, Mass. 1984) 47-57.

[4] Hoel, E. and Samet, H. A qualitative comparison study of data structures for large line segment databases. Proc. ACM SIGMOD Conf., San Diego, CA (1992) 205-214.

[5] Jagadish, H. Linear clustering of objects with multiple attributes. Proc. ACM SIGMOD Conf., Atlantic City, NJ (1990) 332-342.

[6] Lomet, D. Grow and post index trees. Proc. 2nd Symp., SSD 91, Lecture Notes in Computer Science 525 (Springer-Verlag, 1991) 183-206

[7] Lomet, D. and Salzberg, B. Access methods for multiversion data. Proc. ACM SIGMOD Conf., Portland, OR (1989) 315-324.

[8] Lomet, D. and Salzberg, B. The hB-tree: a multiattribute indexing method with good guaranteed performance. ACM Trans. Database Systems 15,4 (Dec. 1990) 625-658.

[9] Orenstein, J. and Merrett, T. A class of data structures for associative searching. Proc. ACM PODS Conf., Waterloo, Canada (1984) 181-190.

[10] Orenstein, J. and Manola, F. PROBE spatial data modeling and query processing in an image database application. IEEE Trans. Software Eng., 14, 5 (1988).

[11] Orenstein, J. Redundancy in spatial database. Proc. ACM SIGMOD Conf., Portland, OR (1989).

[12] Orenstein, J. A comparison of spatial query processing techniques for native and parameter spaces. Proc. ACM SIGMOD Conf., Atlantic City, NJ (1990) 343-352.

[13] Robinson, J. The K-D-B-tree: a search structure for large multi-dimensional dynamic indexes. Proc. ACM SIGMOD Conf. Boston, MA (1984) 10-18.

[14] Sellis, T., Roussopoulos, N., and Faloutsos, C. The R+-tree: a dynamic index for multi-dimensional objects. Proc. VLDB Conf., Brighton, England (1987)

[15] Stonebraker, M. and Frew, J. The SEQUOIA 2000 Benchmark. SEQUOIA 2000 Technical Report No 9, Electronics Research Lab U. of California, Berkeley (March 1992).