Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

A Multidimensional Binary Search Tree for Star Catalog Correlations

D. Nguyen, K. DuPrie and P. Zografou

Smithsonian Astrophysical Observatory, Cambridge, MA 02138

Abstract. A multi-dimensional binary search tree, k-d tree, is proposed to support range queries in multi-key data sets. The algorithm can been used to solve a near neighbors problem for objects in a 19 million entries star catalog, and can be used to locate radio sources in cluster candidates for the X-ray/SZ Hubble constant measurement technique. Objects are correlated by proximity in a RA-Dec range and by magnitude. Both RA and Dec have errors that must be taken into account. This paper will present the k-d tree design and its application to the star catalog problem.

1. Problem Description

One of the requirements for creating the Guide and Aspect Star Catalog for the AXAF (Advanced X-ray Astrophysics Facility) project was to calculate a number of spoiler codes for all objects in the catalog. One of these spoiler codes was calculated based on the distance between an object and its nearest neighbor, and on their magnitude difference. Another group of spoiler codes depended on finding the magnitude difference between an object and the brightest object within a specified radius, where the radius ranged from 62.7 arcmin to 376.2 arcmin. These calculations had to be performed on all 19 million objects in the catalog in a reasonable amount of time. Each object had 14 attributes, taking up 75 bytes of space, resulting in a size of approximately 1.5 GB for the entire catalog. In order to solve this problem we needed a way to search the data by RA, Dec and magnitude fast so repeated searches for each object in the database would be possible within our time limitations. Two different options were immediately available none of which however was ideal.

The first option would use the format in which the catalog is distributed and accessed by a number of applications with relatively simple data needs. This is 9537 *grid* FITS files, each covering a small region of the sky. Since all objects within a file are close to each other in RA and Dec, it is possible to do most positional calculations on a file-by-file basis. In our case, where we were concerned with near neighbors, it would be necessary to deal with a number of neighboring files at at time: the central file whose objects we were calculating near neighbors for, and the surrounding files so we could be sure to find all the near neighbors out to the largest radius. By dealing with a small subset of the data at a time it is possible to load the data in memory but one still has to scan all records in memory until a match is found. This is a time consuming process for the amount of data in the catalog.

Another option was to use a relational database which provides a structure much easier to search than a flat file. The limitation here was the number of dimensions in the search. Although one may create a number of indexes on different fields, including multi-dimensional, any given search is really only using the index in one dimension and scans in all other dimensions. These disk based scans were time consuming and resulted in a performance which was unacceptable. It became clear that a preferably memory based true multi-dimensional search structure was needed.

2. Associative Search

An orthogonal range query for records with multiple keys, say k number of keys, such as the problem described above, is commonly known as *associative search*. In its general form, this problem can formally be described as : the data structure of a record is a (k+1)-tuple, (key_1,key_2,..,key_k, I) where key_i is one of the k key components and I is the additional information field of the record. In the problem above, key_1 is RA, key_2 is Dec, key_3 is magnitude and I refers to any other fields that need to be modified as a result of the calculation. An orthogonal range query is to find all objects which satisfy the condition

$$l_i \leq key_i \leq u_i \tag{1}$$

for i = 1, ... k, where l_i and u_i are the lower and upper ranges of key key_i. A variety of methods have been proposed for solving the multiple keys access problems, unfortunately there is no particular method which is ideal for all applications. The choices for the problem at hand were narrowed down to two : a Multidimensional k-ary Search Tree (MKST) and a k-dimensional (k-d) tree.

2.1. Multidimensional k-ary Search Tree

A MKST is a k-ary search tree generalized for k-dimensional search space in which each non-terminal node has 2^k descendants. Each non-terminal node partitions the records into 2^k subtrees according to the k comparisons of the keys. Note that at each level of the tree, k comparisons must be made and since there are 2^k possible outcomes of the comparisons, each node must have 2^k pointers. The number of child pointers grow exponentially as a function of the dimension of the search space which is a waste of space since many child pointers usually remain unused. An MKST in two (k=2) and three (k=3) dimensional search space is called a quadtree and octree, respectively.

2.2. k-d Tree

A k-d tree is a binary search tree generalized for a k-dimensional search space. Each node of the k-d tree contains only two child pointers, i.e., the size of the node is independent of the number of the dimensional search space. Each non-terminal node of the k-d tree splits its subtree by cycling through the k keys of the k-dimensional search space. For example, the node of the left subtree of the root has a record with value for key key_1 less or equal to value for key key_1 for the root; the node of the right subtree of the root has a record with value for key key_1 greater than the value for key key_1 for the root. The nodes at depth

one partition their subtrees depending on the value for the key key_2. In general, nodes at depth h are split according to $key_{\ h\ mod\ k}$. Note that at each level of the tree only one comparison is necessary to determine the child node. All the attributes of the stars are always known in advanced hence a balanced k-d tree, called an *optimized k-d tree*, can be built. This is done by recursively inserting the median of the existing set of data for the applicable discriminator as the root of the sub tree. An optimized k-d tree can be built in time of O($n\ log\ n$), where n is the number of nodes in the tree. A range search in a k-d tree with n nodes takes time O(m + k $n^(1 - 1/k)$) to find m elements in the range.

3. k-d Tree vs k-ary Search Tree

The k-d tree and the MKST do not exhibit any efficient algorithms for maintaining tree balance under dynamic conditions. For the problem at hand, node deletion and tree balancing are not necessary. The k-d tree was chosen over the MKST for the simple reason that it is spatially (2^k vs 2 child pointers per node) and computationally (k vs 1 comparisons at each level of the tree) more efficient to the MKST as the dimension k increases. The advantage of the MKST over the k-d tree is that the code is slightly less complicated to write.

4. Data Volume Limitation

If the k number of keys to be searched for contain RA, Dec and some other attributes then one can minimize the number of nodes in the tree by utilizing only the near neighbor FITS files. If RA and Dec are not a subset of the k number of keys to be searched then the grid FITS files cannot be utilized and all the stars must be inserted in the nodes of the tree. This could potentially be a problem since the size of the catalog is 1.5 GB. The solution of this problem is to store the k number of keys and the tree structure of the internal nodes in main memory subject to the size of available memory, while the information fields of each star can reside on disk. The disk based informational fields can be linked with the tree structure with the *mmap* utility.

Acknowledgments. We acknowledge support from NASA through grant NAGW-3825.

References

Finkel, R. A. & J. L. Bentley, 1974, "Quadtrees: A data structure for retrieval on composite keys," *Acta Informatica* **4**, 1

J. L. Bentley, 1979, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM* **19**, 509

J. L. Bentley, 1979, "Multidimensional Binary Search Trees in Database Applications," *IEEE Transactions on Software Engineering* **SE-5**, 333

H. Samet, 1984, "The Quadtree and Related Hierarchical Data Structures," *Computing Surveys **16**, 187*