An Empirical Performance Comparison of Some Variations of the $k$-$d$ Tree and $BD$ Tree

Sivarama P. Dandamudi$^1$ and Paul G. Sorenson$^2$

Received July 1984; revised April 1985

Database applications very often require a sophisticated class of storage structures in order to answer different types of queries efficiently. This often dictates that the file should be organized on multiple keys. Several storage structures have been proposed to satisfy these needs. Most of these are a generalization of the storage structures used for managing one-dimensional data. The $k$-$d$ tree is one such example and it is a natural generalization of the standard one-dimensional binary search tree. Recently, a new storage structure, called the $BD$ tree, was proposed to manage multidimensional data. This structure has good dynamic characteristics. Several variations are possible on the basic $k$-$d$ tree structure. This paper studies the performance implications of three variations. Further, it provides an empirical performance comparison of the $k$-$d$ tree and $BD$ tree in database applications.

KEY WORDS: Multidimensional data structures; databases; partial match query; range query; multikey searching.

1. INTRODUCTION

Database systems are becoming increasingly popular as the volume of data that these systems must handle is expanding rapidly. It is now imperative that these database systems should use more efficient storage structures than the traditional ones like indexed sequential files. These traditional

$^1$ Department of Computer Science, University of Maryland, College Park, Maryland 20742.
$^2$ Department of Computational Science, University of Saskatchewan, Saskatoon, Saskatchewan, S7N 0W0, Canada.
storage structures are efficient in answering queries which specify all key values in the record—queries commonly referred to as "exact match" queries. However, an information system must be capable of responding efficiently to various types of queries. Rivest\(^{(1)}\) classified the queries in a database environment into several categories. As far as this paper is concerned, we are interested in the following three types of queries.

(a) **Exact match query:** This is the simplest type of query and specifies a value for each key in the record.

(b) **Partial match query:** Assuming that each record contains \(k\) keys, then a partial match query specifies \(s\) key values, \(s < k\), that must be matched. The remaining \(k - s\) keys are left unspecified.

(c) **Range query:** These are the same as exact match queries except that a range of required values rather than a single value may be specified for each key.

In a way, the first two types of queries can be considered as a subset of the range queries. For example, in an exact-match query, the range of all attributes can be thought of as one (some specified value). Similarly, we can consider the range of those attributes that are not specified in a partial match query to be \(-\infty\) to \(+\infty\). Here \(+\infty\) indicates the maximum possible value for the attribute and \(-\infty\) indicates the minimum possible value.

The traditional storage structures are not efficient in answering different types of queries and, therefore, most database applications now require a more sophisticated class of storage structures—called multikey storage structures. As a result of this need and interest, several new multikey access storage structures have been reported in the literature. Bentley and Shamos\(^{(2)}\) consider the problem of ranking a point in a multidimensional space. They developed a data structure, called ECDF tree, to solve this problem in \(O(N \cdot (\log N)^k)\) time using \(O(N \cdot (\log N)^{k-1})\) space. Bentley\(^{(3)}\) has investigated a number of searching problems defined on sets of points in \(k\)-dimensional space. It should be noted that achieving a lower bound for range queries (see Ref. 4) requires excessive amounts of storage. Some efficient worst-case data structures have been reported in Ref. 5. Since exact match and partial match queries can be considered as a subset of range queries, several data structures have been proposed to handle range queries efficiently (see Refs. 6 and 7). Some of these highly "efficient" data structures like quintary trees\(^{(8)}\) provide a very good response time. However, these structures are not suitable for most applications because of their huge storage requirement. There are several other storage structures that are more generally suitable for implementation because they provide a reasonable compromise between the storage cost and response time. These