Reduced Memory Space for Multi-Dimensional Search Trees (Extended Abstract)

Dan E. Willard

State University of New York at Albany, Albany, New York 12222 and consultant to Bell Communications Research

Abstract

Our main result is that space $O[N(log N / log log N)^{k-1}]$ suffices for either doing dynamic k-dimensional aggregate orthogonal range queries in time $O(log^k N)$ on a set of $N$ records, or for arbitrary $\varepsilon > 0$ to do static aggregate queries in time $O(log^{k-1+\varepsilon} N)$. This result improves upon the memory space used by slightly more than one dozen previous authors by a factor $O[(log log N)^{k-1}]$, and it has applications to rectangle intersection problems, VLSI-design, relational data bases, and queries about the past.

1. Introduction

Throughout this paper, $S$ will denote a set of $N$ points in $k$-dimensional space. An orthogonal range query of dimension $k$ is defined as a request for the subset of $S$ that satisfies an inequality similar to

$$a_1 < KEY.1 < b_1 \& a_2 < KEY.2 < b_2 \& \ldots \& a_k < KEY.k < b_k.$$  \hspace{1cm} (1.1)

Often $q$ will denote an orthogonal range query, SET$(q)$ the set of records satisfying $q$ and COUNT$(q)$ the size of this set.

Suppose each record $R$ in $k$-dimensional space has an attribute VALUE$(R)$ that belongs to some abelian group $G$. Then a request to calculate $\sum_{R \in SET(q)}$ VALUE$(R)$ is called a $k$-dimensional aggregate query. A large number of articles [AS81, Be80, DM80, Fr81, EO81, LW80, LW82, Me84, Wi78a, Wi78b, Wi85, WL85] starting with [BS77] have studied a variety of problems about data structures occupying space $O(N log^{k-1} N)$ and permitting $k$-dimensional aggregate orthogonal range queries to run in dynamic time $O(log^k N)$ and permitting static aggregates to run in time $O(log^{k-1} N)$ [Wi78b]. Our main goal will be to reduce the memory space of these data structures by a factor of $log^{k-1} N$ and show that space $O[N(log N / log log N)^{k-1}]$ provides the facility to do static K-dimensional aggregate queries in time $O(log^{k-1+\varepsilon} N)$, or to do dynamic operations in time $O(log^k N)$.

Sometimes, a user will not wish his algorithm to calculate an aggregate but rather to compute the full list of records satisfying equation (1.1). This variant of the orthogonal query is called the set-enumeration query, and a data structure is said to have a locate time $O(F)$ iff every set-enumeration query listing COUNT$(q)$ distinct elements runs in time $O(F + COUNT(q))$.

Locate retrieval time should not be confused with aggregate retrieval complexity. Chazelle [Ch83] has shown that space $O[N log^{k-1} N / log log N]$ makes possible locate retrieval complexity $O(log^{k-1} N)$. The two main differences between Chazelle's result and ours are that
the two articles study different types of retrieval operations and that [Ch83] achieves a better time while we achieve a better space. A curious fact is that there is no apparent method to generalize [Ch83]'s algorithm to aggregate retrievals or sections 3 and 4 of our article to locates. This striking difference between the aggregate and locate search problem is not without precedent; the contrast between Fredman's lower bound $\Omega(\log^2 N)$ for the combined worst-case cost for insertions, deletions and aggregate retrievals [Fr81] and Willard's upper bound $O(\log^{k-1/2} N)$ for the analogous k-dimensional locate problem [Wi85] provides at least one documented example where it is known that these complexities will never match.

The main discussion in sections 2 thru 4 of this paper concerns aggregate retrievals, but section 5 will veer away from the main subject and outline a dynamization technique which enables Chazelle's data structure to accommodate $O(\log^2 N / \log \log N)$ insertions and deletions. Yao has proven [Ya82] a time-space lower bound, and the main open questions raised by our article concerns the unexplored gap between our upper bounds and Yao's lower bounds. Section 6 discusses the application of our theorem to detecting rectangle intersection, relational data bases and queries about the past.

2. Literature Survey

The term first-generation k-fold tree refers to a data structure that was first proposed in [BS77] and has subsequently been discussed in [AS81, Be80, DM80, EO81, Fr81, LW80, LW82, Me84, Wi78a, WL85]. It occupies space $O(N \log^{k-1} N)$ and was shown by [BS77] to support $\log^2 N$ aggregate and locate retrieval time. Lueker and Willard [LW82, WL85] have shown this data structure also supports $\log^2 N$ update time, and Fredman showed this result had optimal time for the aggregate model of computation [Fr81]. As early as 1978, Willard had shown it was possible to obtain certain types of $\log N$ improvements not violating Fredman's lower bound [Wi78b, Wi85], and more recently [Ch83, Ed81] outlined related types of improvements and [GBT84] discussed geometric scaling problems. Each of these new data structures as well as the new variants in this paper use concepts from first-generation k-fold tree theory. It is therefore appropriate to use the term second-generation k-fold trees for such alternate data structures.

The symbol $T_1(k, S)$ will denote a first generation k-fold tree that represents a set $S$, and the two main new data structures devised in this paper will be called $T_{fg}(k, S)$ and $T_{fg'}(k, S)$.

Sometimes for simplicity, we will omit the symbols $k$ and $S$ when discussing our new data structure; that is, $T_1$ and $T_f(k)$ will sometimes serve as abbreviations for $T_1(k, S)$. Before defining our new data structures, it is useful to review the definition of the first-generation data structure $T_1$. This review, as well as most of the other discussion in this paper, will take place in the context of the dimension $k=2$; this dimension provides both the simplest example of nontrivial k-fold trees and their most practical example.