The K-Vector ND and its Application to Building a Non-Dimensional Star Identification Catalog

Benjamin B. Spratling IV and Daniele Mortari

Abstract

A multi-dimensional orthogonal range-searching algorithm, the multi-dimensional k-vector (K-Vector ND), is presented. The algorithm is analyzed and found to have an execution time that is independent of the size of the database, for well-distributed data sets. Numerical tests are performed to determine the performance advantage as compared to a Quad-Tree for the two-dimensional data set. Results range from break-even to a factor of 14, depending on the database size. The K-Vector ND is then applied to the problem of building a non-dimensional star-identification database that contains all visible star triples. The performance of the K-Vector ND algorithm in that task is then compared to a simple nested loop, and found to range from break-even to a factor of 200, depending on the size of the database.

Introduction

A traditional aerospace engineer tends to have a measure-and-move-on approach when working with algorithms. If an engineer knows two algorithms that both get the job done, and does not know which one is faster, he might code both, run a sample data set on them, and then select the faster algorithm. Although this method is correct in the sense that it actually finds the fastest algorithm for the given data set, it fails to take into account how the run-times of the algorithms vary by the size of the data set, which has a large impact on applications that must search a database. If one were to double the size of the data set, one algorithm might take four or eight times longer, whereas the other might only take double the time. A
large enough data set can make a relatively slower algorithm relatively faster. If the wrong algorithm is chosen, it can significantly hamper performance, especially on the constrained platforms typically available for aerospace applications.

The difference in the relative performance for different algorithms is directly related to how the algorithms operate. Computer scientists analyze the running time of algorithms using “asymptotic” notation, which is similar to the way an engineer would notate higher order terms in an infinitesimal analysis that are too small to bother to use. For more information about asymptotic notation both $O()$ and $\Theta()$, see Chapter 3 of *Introduction to Algorithms* [1].

The asymptotic running time of an algorithm may include factors other than the size of the database. It might be based on any value from the data set or a query. For example, if an algorithm attempts to find all the numbers in a database between two given values, and there are $k$ correct answers, the algorithm’s running time might be proportional to $k$, for instance, $O(n + k)$. For this article, we will also use $d$ to represent the number of dimensions, $f$ to represent the number of stars in a field-of-view (FOV). For the engineer, algorithms with better asymptotic performance will eventually perform faster as data sets continue to grow.

A “Range Search” finds all the items in a database between two values. In the multidimensional sense, in which each entry in a database contains a value along each axis, an “orthogonal” search would find all entries in the database for which the value in each dimension lies between the two given values for that dimension. As an example, consider a two-dimensional database composed of the declination and right-ascension of stars.

**k-Vector**

There are many ways to search a database, and the previous section described a “linear” search, a method in which the computer examines every item in the database. There are other methods for performing this search, if the data has been organized in some way. For instance, if the data has been sorted from lowest to greatest, then searching for a particular value can avoid looking at half the data values by checking the entry in the middle of the data set. If the value is too low, the algorithm can check the top half, and if it is too high, then it can check the lower half. This particular technique is called a “binary search,” because at each interval it checks whether to look at the top or bottom half. By the end of the search the algorithm has looked at $\log_2 n$ entries, so the asymptotic running time is listed as $O(\log_2 n)$.

There are a few caveats to the advertised performance analysis, as will be discussed in the conclusions, but the first is the type of search that is performed. The quad-tree and other linear searches will compare the data values of the entries with the requested ranges; all entries lying inside the ranges are found, and no entries lying outside the search range are found. The algorithm herein, however, will find some results that are slightly outside the search range. Although the algorithm can easily be modified to filter these entries, many orthogonal search ranges are merely circumscribed bounds to a non-orthogonal region, such as a circle. In such cases it is not efficient to use the rectangular values to reject entries because other nonlinear criteria may more efficiently reject entries that lie in the orthogonal search ranges.

The basic idea behind the *k*-vector [2–5] is the division of the range of data into “bins” which represent closely related data values that lie within a given, standard