# Chromatic Nearest Neighbor Searching: A Query Sensitive Approach

David M. Mount* Nathan S. Netanyahu† Ruth Silverman‡ Angela Wu§

## Abstract

The nearest neighbor problem is that of preprocessing a set $P$ of $n$ data points in $R^d$ so that, given any query point $q$, the closest point in $P$ to $q$ can be determined efficiently. In the *chromatic* nearest neighbor problem, each point of $P$ is assigned a color, and the problem is to determine the *color* of the nearest point to the query point. More generally, given $k \geq 1$, the problem is to determine the color occurring most frequently among the $k$ nearest neighbors. The chromatic version of the nearest neighbor problem is used in many applications in pattern recognition and learning. In this paper we present an algorithm for solving the chromatic $k$ nearest neighbor problem. Although in the worst-case, our algorithm runs no faster than the best nonchromatic version, we provide a *query sensitive* analysis, which shows that if the color classes form spatially well separated clusters (as often happens in practice) then queries can be answered more efficiently. We present our results in the more general context of approximate nearest neighbor searching, in which the user provides a relative error, $\epsilon \geq 0$, and the algorithm can answer the query with respect to any $k$ approximate nearest neighbors. We present empirical evidence that for well clustered data sets, this approach leads to significant improvements in efficiency.

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. The support of the National Science Foundation under grant CCR–9310705 is gratefully acknowledged.

†Center for Automation Research, University of Maryland, College Park, and Space Data and Computing Division, NASA Goddard Space Flight Center, Greenbelt, Maryland. This research was carried out, in part, while the author held a National Research Council NASA Goddard Associateship.

‡Department of Computer Science, University of the District of Columbia, Washington, DC, and Center for Automation Research, University of Maryland, College Park, Maryland.

§Department of Computer Science and Information Systems, The American University, Washington, DC.

## 1 Introduction.

Let $P$ denote a set of points in $d$-dimensional space. Given a query point $q \in R^d$, the *nearest neighbor* to $q$ is the point of $P$ that is closest to $q$ under any Minkowski distance metric. These metrics include the well-known Euclidean distance, rectilinear or Manhattan distance, and the max-norm. Computing nearest neighbors in moderately high dimensional spaces is a flexible and important geometric query problem with numerous applications in areas such as pattern recognition, learning, statistics, and data compression.

In many of these applications, particularly those arising from pattern recognition and learning, the set $P$ is partitioned into $c$ disjoint subsets, $\{P_1, P_2, \ldots, P_c\}$, called *color classes* (or *patterns*), and the problem is to determine the index of the color class of the nearest neighbor to $q$. No information need be given as to which point in $P$ realizes this color. More generally, to improve the robustness of classification, for some $k \geq 1$, the problem is to determine which color occurs most frequently (i.e. the mode) of the $k$ nearest neighbors. The *chromatic k-nearest neighbor problem* is that of preprocessing the set $P$ so that chromatic

nearest neighbor queries can be answered efficiently. For the applications in mind, we can take $k$ and $d$ to be fixed independent of $n$.

There are other ways to formulate such a query [8, 9]. (For example, another formulation is to compute the number of distinct colors occurring among the $k$ nearest neighbors). Our choice is motivated from classification applications in pattern recognition, learning, and image processing.

In this paper we consider more generally an approximate version of the chromatic nearest neighbor and $k$-nearest neighbor problems. Given a user supplied parameter, $\epsilon \geq 0$, a $(1+\epsilon)$-*nearest neighbor* of a query point $q$ is defined to be a $p \in S$, such that, if $p^*$ is the closest point in $P$ to $q$, then

$$\frac{dist(q,p)}{dist(q,p^*)} \leq 1+\epsilon.$$

In other words, $p$ is within relative error $\epsilon$ of being a nearest neighbor of $q$. We say that color class $i$ is a $(1+\epsilon)$-*chromatic nearest neighbor* of $q$ if there is a point of color $i$ that is a $(1+\epsilon)$-nearest neighbor of $q$.

For the generalization to $k$ approximate nearest neighbors, define a sequence of $k$ *approximate nearest neighbors* to be any sequence of $k$ *distinct* points from $P$, $\langle p_1, p_2, \ldots, p_k \rangle$, such that for $1 \leq i \leq k$, $p_i$ is within a relative error $\epsilon$ from the true $i$-th nearest neighbor of $q$. Clearly, there may be many different approximate nearest neighbor sequences for a given query point. The approximate version of the chromatic $k$-nearest neighbors problem is that of determining the color occurring most frequently among any sequence of $k$ approximate nearest neighbors. Throughout we assume that the parameters $\epsilon$ and $k$ are supplied as a part of the query, and are not known at the time of preprocessing.

The well known standard (nonchromatic) nearest neighbor problem can be thought of as a special case of the chromatic problem, where every point has its own color. Algorithms and data structures for the standard nearest neighbor problem have been extensively studied [5, 6, 11]. For existing approaches, as the dimension grows, the complexity of answering exact nearest neighbor queries increases rapidly, either in query time or in the space of the data structure used to answer queries. However, Arya et al. [3] showed that *approximate* nearest neighbor queries can be answered quite efficiently. They established that after $O(n \log n)$ preprocessing, a data structure of size $O(n)$ can be built, such that nearest neighbor queries can be answered in $O(\log n)$ time. Constant factors in query time are on the order of $(a/\epsilon)^d$, for some constant $a$, and constant factors in space are on the order of $d$. Recently, Clarkson [7] has presented an alternative approach in which the constant factors show a lower dependence on the dimension.

In this paper we present an approach for the approximate chromatic nearest neighbor problem which is sensitive to clustering in data sets. We analyze its performance not from a worst-case perspective, but from a *query sensitive* perspective. A query sensitive analysis is one which describes the running time of an algorithm as a function not only of input size, but of a set of one or more parameters, which are intrinsic to the geometry of the query. In general, these parameters should capture, in a relatively intuitive way, the underlying complexity of answering queries. Ideally, such an analysis should show the algorithm takes advantage of simplifying factors to improve running times.

For the chromatic $k$ nearest neighbor problem we introduce the following geometric parameters. These quantities are based on the assumption that points have been scaled to lie within a unit hypercube. Let $r_k$ denote the distance from the query point $q$ to its $k$-th nearest neighbor. Let $r_\chi$ denote the largest radius such that the mode color among the $k$ nearest neighbors is determined purely from the cardinalities of colors appearing within this distance of $q$. For example, $r_1$ is the radius of the largest monochromatic ball centered at $q$. For general $k$ and $r \geq 0$, let $M(r) = \langle m_1(r), \ldots, m_c(r) \rangle$ denote the nonincreasing sequence of color cardinalities among the points in in a ball of radius $r$ centered at $q$. (Thus, $m_1(r)$ is the number of times the most frequent color occurs up to distance $r$.) Among the $k$ nearest neighbors at least $k - \sum_{i>1} m_i(r)$ are of the most frequently occurring color. This color will be the mode if this quantity is at least as large as the cardinality of the second most frequent color. Thus,

define $r_\chi$ be the largest $r \geq r_k$ such that

$$k - \sum_{i>1} m_i(r) \geq m_2(r).$$

Note that $r_\chi > r_k$, since for $r = r_k$ the left hand side of this inequality is equal to $m_1(r)$, which is at least as large as $m_2(r)$. Also observe that when $k = 1$, the above inequality is satisfied if and only if the ball of radius $r_\chi$ is monochromatic, and hence this generalizes the definition for single nearest neighbor case.

Based on these values, we introduce the following parameters upon which our analysis is based. (The titles associated with these parameters are intended to be descriptive rather than definitions.)

**Cluster size:** is defined to be $r_\chi$. Since the diameter of the point set is fixed, the larger the cluster of points containing $q$, the larger we expect $r_\chi$ to be. Intuitively, if $q$ lies within the middle of a large cluster of points of the same color, then it should be easier to determine the color the nearest point to $q$.

**Local density:** Let $\delta = (r_\chi - r_k)/r_k$. Clearly $\delta \geq 0$. This parameter plays a role similar to $\epsilon$ by bounding the precision with which the query needs to be answered.

We believe that these parameters intuitively capture the strength of the clustering near the query point (as these parameters become larger, query processing should become easier). Indeed, we show that after $O(n \log n)$ preprocessing and with $O(n)$ space, chromatic approximate nearest neighbor queries can be answered in time

$$O\left(\min\left(\log n,\ \log \frac{1}{r_k \cdot \max(\delta, \epsilon)}\right)\right),$$

with constant factors growing roughly as $O(kc(\sqrt{d}/\max(\delta,\epsilon))^d)$. Observe that the worst-case running time is as good as the nonchromatic algorithm by Arya, et al. [3] (for $k = 1$), but if $\delta$ is large, then the running time may be much better. When $\delta > \epsilon$, the running time is independent of $\epsilon$, and the algorithm will return the correct color in this case. The constant factors in space and preprocessing time are $O(cd)$. The fact that exponential constant factors do not appear in the space complexity is an important practical consideration for applications in higher dimensions.

# 2 Search Algorithm.

Recall that the problem is that of determining the color occurring most frequently (the mode) among any set of $k$-approximate nearest neighbors to the query point. Note that no information (exact or approximate) on the distance to the $k$ nearest neighbor need be supplied. Preprocessing consists of building a balanced box decomposition tree (BBD tree) for the data points. This is a quadtree-like data structure of height $O(\log n)$ that is based on a decomposition of space into rectangular-based *cells*, each of which is of bounded *aspect ratio* (they are not too skinny). See [4] for a more complete description of this data structure. Recall that $r_k$ denotes the (unknown) distance from the query point to its $k$-th nearest neighbor, and that $r_\chi$ is the largest ball from which the mode color can be inferred purely from color counts (defined in the introduction).

The algorithm maintains a pair of bounding radii $r^-$ and $r^+$ such that throughout the search,

$$r^- \leq r_k \leq r^+.$$

As the algorithm proceeds, the lower bound $r^-$ increases monotonically from 0, and the upper bound $r^+$ decreases monotonically from $\infty$. The algorithm operates by repeatedly expanding the unexpanded nodes of largest size, and updating estimates for $r^+$ and $r^-$. For each node we assume we know the color counts for points in the corresponding cell, but we know nothing about the locations of the points within this cell. Termination occurs either when we can infer the mode color from the current color counts, or when the cells have been expanded to such a small size that we can infer the $k$ approximate nearest neighbors irrespective of their specific locations within their cells.

To minimize the number of nodes that participate in the computation at any time, if it can be inferred that a cell associated with a given node lies completely outside or completely inside the ball of radius $r_k$, this node

need not be expanded further. For each node $v$, let $dist^-(v)$ ($dist^+(v)$) denote the minimum (maximum) distance from the query point $q$ to the cell associated with $v$. (These distances can be computed in $O(d)$ time from $BB(v)$ and $IB(v)$.) The algorithm maintains three groups of nodes, *In*, *Out*, and *Mid*, such that

$$v \in In \Rightarrow dist^+(v) \leq r^-,$$
$$v \in Out \Rightarrow dist^-(v) \geq r^+.$$

Nodes which satisfy neither of these conditions are placed in the third group, *Mid*.

Every point associated with an *In*-node lies within the ball of radius $r_k$, and every point in associated with an *Out*-node lies outside this ball. At all times, the union of the cells associated with these three groups of nodes forms a subdivision of the bounding hypercube for the data points into regions with pairwise disjoint interiors, and hence induces a partition on the point set. Here is a description of the algorithm.

(1) (Initialization) $r^- = 0$, $r^+ = \infty$, $Mid = \{root\}$, and $In = Out = \emptyset$.

(2) Repeat the following steps.

(a) (Expansion) Let $s$ be the largest sized node in *Mid*. For each node $v$ in *Mid* of size $s$, replace $v$ by its children in the BBD tree, adding each to *Mid*. (If the children are also of size $s$, then expand them as well.)

(b) (Update distances) For each new node $v$ in *Mid* compute $dist^-(v)$ and $dist^+(v)$. For the node $v \in In \cup Mid$ associate this node with $dist^-(v)$, and weight $weight(v)$. Using a weighted selection algorithm, let $r^-$ be the element of rank $k$ in this list. Compute $r^+$ similarly, but using $dist^+(v)$.

(c) (Classification) For each node $v \in Mid$:

- if $(dist^+(v) \leq r^-)$ then add $v$ to *In*.
- if $(dist^-(v) > r^+)$ then add $v$ to *Out*.
- otherwise leave $v$ in *Mid*.

(d) (Termination condition 1) Sum the color counts for all points in $In \cup Mid$. Let $M = \langle m_1, \ldots, m_c \rangle$ denote the nonincreasing sequence of these colors. If

$$k - \sum_{i>1} m_i \geq m_2,$$

then return the color of $m_1$ as the answer.

(e) (Termination condition 2) If $4s\sqrt{d} \leq \epsilon r^-$ then enumerate the $k$ witnesses to $r^-$, and return the mode color of this set.

This algorithm is dovetailed with the $k$ nearest neighbor algorithm presented by Arya, et al. [3]. The reason is that if the points are not well-clustered, then we want to be able to guarantee a maximum $O(k \log n)$ time bound.

The correctness of the algorithm is straightforward to establish, and details have been left to the full paper. One issue which was not addressed in the algorithm is how nodes are selected for expansion. The method is analogous to the one described in [4] and is omitted from this version.

# 3 Query Sensitive Analysis.

In this section we prove that the search algorithm described in the previous section terminates within time

$$O\left(\log \frac{1}{r_k \cdot \max(\delta, \epsilon)}\right),$$

where $\delta$ is the query sensitive parameter defined in the introduction. The algorithm expands nodes in stages in groups according to their size. Thus, at stage $j \geq 1$ the expanded nodes are of size $s_j = 1/2^j$. Thus the diameter of each expanded cell at stage $j$ is at most $s_j\sqrt{d}$. First, we observe that the estimates $r^-$ and $r^+$ are accurate estimates of $r_k$ to within this value. (Proofs are omitted from this version.)

**Lemma 3.1** *Immediately after the completion of stage $j$,*

$$r^- \geq r_k - s_j\sqrt{d} \quad \text{and} \quad r^+ \leq r_k + s_j\sqrt{d}.$$

**Lemma 3.2** *For all sufficiently small $\epsilon$, the algorithm terminates as soon as*

$$s_j\sqrt{d} \leq r_k \max\left(\delta, \frac{\epsilon}{8}\right).$$

Because the size of cells decreases by a factor of 1/2 for every constant number of levels of descent in the tree, it follows immediately that the number of stages until termination is

$$O\left(\log \frac{1}{r_k \cdot \max(\delta, \epsilon)}\right).$$

Next, we claim that at each stage of the algorithm only a constant number of cells can be expanded, depending on $d$ and $\epsilon$. All of these cells must overlap a ball of radius $r^+$ centered at $q$ (otherwise they will not be expanded). It suffices to consider the lowest level of expansion, since this will have the smallest cell size, and hence require the largest number of cells to cover the ball of radius $r^+$. Lemma 3.2 specifies the smallest sized node that we consider at this level, and using the fact that the expanded cells are pairwise disjoint from one another and of bounded aspect ratio, by applying a packing argument, it follows that the number of cells that overlap the ball of radius $r^+$ is

$$O\left(\left(1 + \frac{\sqrt{d}}{\max(\delta, \epsilon/8)}\right)^d\right),$$

which is $O(1)$ for fixed $d$ and $\epsilon$. By the arguments made at the end of the previous section, each stage requires time proportional to the number of active nodes together with an additional factor of $c$ to process each stage. This establishes the running time stated at the beginning of this section.

# 4 Experimental Results

To establish its efficiency empirically, we implemented a variation of this algorithm in C++ (called, *cann*, for chromatic approximate nearest neighbors), and we compared its performance to the (nonchromatic) $k$ approximate nearest neighbor algorithm developed by Arya, et al. [3] (called *ann*). Both algorithms construct essentially the same tree (although the tree of [3] contains no chromatic information). This tree is a binary variant of the BBD tree. Details are presented in the full paper.

In both cases, splitting stopped when four points or fewer resided in a box. The ann algorithm computed the $k$ approximate nearest neighbors, and then returned the mode color of these points. In all cases we used $k = 5$ as the number of nearest neighbors.

For each algorithm, we measured a number of quantities for each run: the number of tree nodes visited by the search, the number of points encountered, and the number of times a coordinate of a point was encountered.

Our goal was to show that for well clustered data sets, cann outperforms ann, and to investigate the sensitivity of cann to clustering. We ran our algorithm on two general categories of experiments, synthetically generated data sets and real application data sets. Due to space limitations, we only show results on the application data set.

The data points were selected from Landsat-TM5 images. The landsat image data consisted of a collection of pixels, where each pixel is broken down into 7 spectral bands digitized over the range 0 to 255 (and hence each is a point in a 7 dimensional space). The data had already been classified, through an interactive photointerpretation procedure, into 7 different classes, according to the U.S. Geological Survey land-use land-cover (LULC) classification scheme introduced in [1]. From a large data set, we selected 51,609 data entries. Two data sets of size 10,000 were randomly sampled from this file. For this version we show results on the second data set.

Because the data points had already been classified off-line, we measured the performance of both algorithms on a class-by-class basis, generating 500 query points from each class (from a different source than data points), and running each against on of the 10,000-entry data files. We only tested query points whose classes appeared in the data set. Experiments were run with $\epsilon$ values ranging from 0.1 to 10.

In Figure 1 and 2 we show the results of 3 representative groups of queries: barren, forest and water on each of the two data sets. For these plots the $x$-axis is the value of $\epsilon$. As before, solid lines represent the cann algorithm and dashed lines represent the ann al-

gorithm. It can be seen that in the first data set, water, which exhibited the greatest clustering and separation, demonstrated the greatest improvement in running time for cann over ann. In contrast, barren and forest performed more poorly (owing largely to contamination from other overlapping classes). In the second experiment, the removal of some of the contaminating classes resulted in an improved performance for cann.

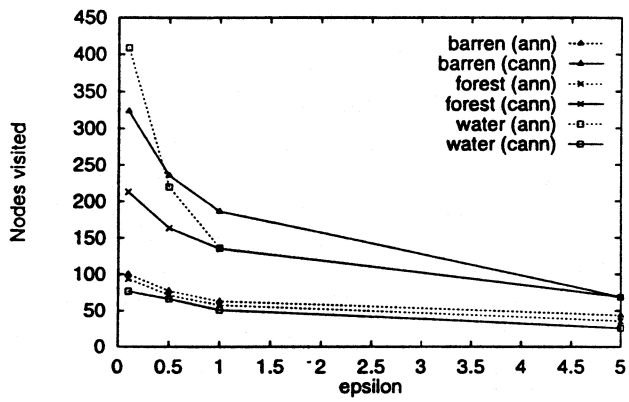


Figure 1: Landsat experiment 1.

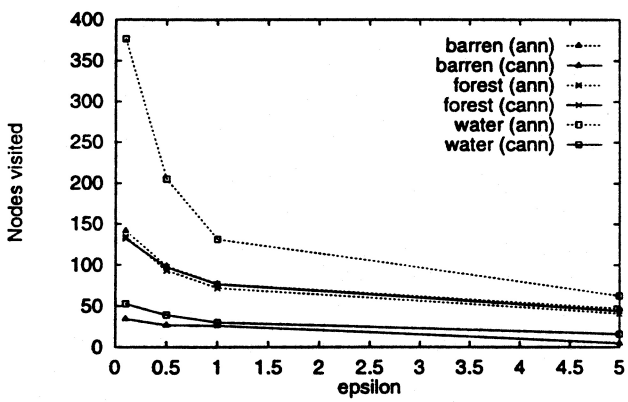


Figure 2: Landsat experiment 2.

In summary, the experiments have shown that for data sets which are well clustered and in which clusters are well separated, cann provides a significant performance improvement over the nonchromatic ann algorithm.

# References


[1] J. R. Andereson, E. E. Hardy, J. T. Roach, and R. E. Witmer. *A Land Use and Land Cover Classification System for Use with Remote Sensor Data*, Geological Survey Professional Paper 964, 1976.

[2] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 573–582, 1994.

[4] S. Arya, D. M. Mount, Approximate Range Searching. *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, to appear.

[5] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Transactions on Mathematical Software*, 6(4):563–580, 1980.

[6] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[7] K. L Clarkson. Algorithms for polytope covering and approximation, and for approximate closest-point queries. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994.

[8] T. Graf and K. Hinrichs. Algorithms for proximity problems on colored point sets. In *Proc. 5th Canad. Conf. Comput. Geom.*, 1993, pages 420–425.

[9] P. Gupta and R. Janardan and M. Smid. Efficient algorithms for generalized intersection searching on non-iso-oriented objects. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pages 369–378.

[10] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[11] R. L. Sproull. Refinements to nearest-neighbor searching in $k$-dimensional trees. *Algorithmica*, 6:579–589, 1991.