Chromatic Nearest Neighbor Searching: A Query Sensitive Approach*

David M. Mount[†] Nathan S. Netanyahu[‡] Ruth Silverman[§]
Angela Y. Wu[¶]

Last modified: October 21, 1996.
(Minor updates made: August 24, 2000.)
Final version for *CGTA*.

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 a simple algorithm for solving the chromatic k nearest neighbor problem. 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 quite efficiently. We also allow the user to specify an error bound $\epsilon \geq 0$,

*A preliminary version of this paper appeared in the *Proceedings of the 7th Canadian Conference on Computational Geometry*, 1995, 261–266.

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Email: mount@cs.umd.edu. The support of the National Science Foundation under grant CCR–9310705 is gratefully acknowledged.

[‡]Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, and Center for Automation Research, University of Maryland, College Park, Maryland 20742. This research was carried out while the author was also affiliated with the Center of Excellence in Space Data and Information Sciences at NASA Goddard Space Flight Center, Greenbelt, MD 20771. Email: nathan@macs.biu.ac.il.

[§]Center for Automation Research, University of Maryland, College Park, MD 20742. Email: ruth@cfar.umd.edu. The support of the National Science Foundation under ROA supplement CCR–9310705 is gratefully acknowledged.

[¶]Department of Computer Science and Information Systems, American University, Washington, DC 20016. Email: awu@american.edu.

and consider the same problem in the context of approximate nearest neighbor searching. We present empirical evidence that for well clustered data sets, this approach leads to significant improvements in efficiency.
Key words: Chromatic nearest neighbors, classification algorithms, pattern recognition, multidimensional searching, BBD trees, branch and bound search, query sensitive analysis.

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. Throughout we assume that distances are measured using any Minkowski metric [29]. These metrics include the familiar Euclidean (L_2) distance, the Manhattan or city-block (L_1) distance, and the max (L_∞) distance. 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 (see e.g., [2, 19, 23, 33]) 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 color of q's nearest neighbor. No information need be given as to which point in P realizes this color. More generally, to improve the robustness of classification, for some constant $k \geq 1$, the problem is to determine which color occurs most frequently (i.e. the mode) among 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 consider d, c, and k to be fixed constants, independent of n.

There are other ways to formulate queries based on colored points. For example, one formulation is the nearest foreign neighbor problem in which the query point is associated with a color and the problem is to find the nearest point of a different color [21]. Another possible formulation is to compute the number of distinct colors occurring among the k nearest neighbors [22]. Our choice is motivated by the following application from pattern recognition and learning. For some fixed d, objects are encoded by a d-element vector of numeric *properties*. Vector lengths in the range from 5 to 20 are quite common. These vectors are interpreted as points in d-dimensional space. In these applications, it is assumed that points are drawn from a population consisting of c different distributions. A large number of points are selected as a *training set*, and these points are partitioned into color classes by an external agent. Given a query point, it is classified by determining the color of the nearest point(s) in the training set.

For example, in an application like optical character recognition, the color classes might be the set of English characters, and the training set is formed by scanning and

digitizing a large set of characters, and then extracting a vector of information from each digitized character. We do not make any claims regarding the soundness of this type of nearest neighbor classification as a method for pattern recognition. However, this approach is used in many existing systems.

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 [4, 8, 10, 15, 16, 17, 28, 31, 32, 35]. 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. The growth rate is so rapid that these methods are not of real practical value in even moderately high dimensions. However, it has been shown that *approximate* nearest neighbor queries can be answered much more efficiently. The user supplies a value $\epsilon \geq 0$ along with the query point, and the algorithm returns a point that is within a factor of $(1 + \epsilon)$ from the true nearest neighbor. Arya et al. showed 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 $O(\log n + (1/\epsilon)^d)$ time (assuming fixed d). Recently, Clarkson [16] has presented an alternative approach in which the constant factors show a lower dependence on the dimension.

The chromatic version of the k-nearest neighbor problem has been studied in applications contexts (see e.g., [20, 25, 26, 27]), but not from the perspective of worst-case asymptotic complexity. One reason is that in the worst case, it is not clear how to determine the most common color among the k nearest neighbors without explicitly computing the k nearest neighbors. However, in many of the applications of chromatic nearest neighbor searching, the problem has been so constructed that the color classes are spatially well-separated. For example, in the pattern recognition application mentioned earlier, a large number of properties (e.g., on the order of hundreds) are chosen initially, and then a heuristic clustering procedure is invoked to determine a subset of properties (e.g., on the order of 10 to 20) that provide the strongest discrimination between classes [24]. Thus color classes are explicitly chosen to create *clusters* that are spatially separated from one another. In these contexts, worst-case complexity may not be a good criterion for evaluating nearest neighbor algorithms, because input instances have been designed to avoid worst-case scenarios. Thus, it is important for good practical performance to consider how to take advantage of clustering.

In this paper we present an approach for the approximate chromatic nearest neighbor problem that is sensitive to clustering in the data set. 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. These parameters should capture, in a relatively intuitive way, the underlying complexity of answering queries. Ideally, such an analysis should

show that the algorithm takes advantage of natural simplifying factors to improve running times.

What is a natural parameter for describing the complexity of k nearest neighbor searching? The single nearest neighbor case is somewhat simpler to understand, so we consider this case first. Given a query point q, to establish the color of the nearest neighbor, let r_1 denote the distance to the nearest neighbor, and let r_χ denote the distance to the nearest point of any other color. Intuitively, if the query point is located within a cluster of points of a given color, then we expect r_χ to be large relative to r_1, or equivalently, $(r_\chi - r_1)/r_1$ should be large. (See Figure 1.) We call this ratio the *chromatic density* about the query point, and denote its value by $\delta(q)$. Clearly if points are in general position then $\delta(q) > 0$. Observe that the color of q's nearest neighbor is determined by knowledge of the colors of data points within distance r_χ, but without knowledge of their relative distances from q.

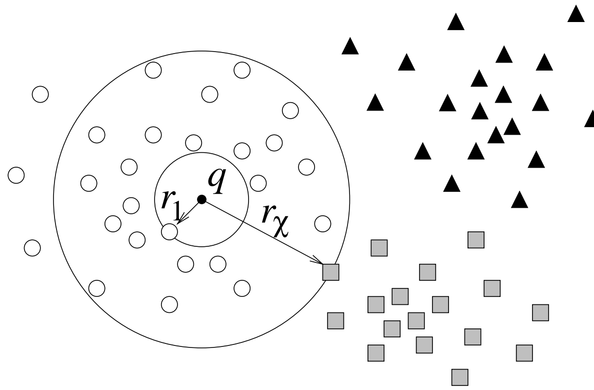

Figure 1: Chromatic Density.

How can this definition be generalized for arbitrary k nearest neighbors? First, let r_k denote the distance from q to its kth nearest neighbor. Intuitively, we define r_χ to be 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 (and, in particular, without knowledge of their relative distances from q). More precisely, for $r \geq 0$, let $M(r) = \langle m_1(r), \ldots, m_c(r) \rangle$ denote the nonincreasing sequence of color cardinalities among the points in a closed ball of radius r centered at q. Thus, $m_1(r)$ is the number of occurrences of the most frequently occurring color within distance r of q. Define the *excess* to be

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

Intuitively, the excess indicates how many more points are of the most common color, compared against the second most common color. The following lemma establishes the connection between $Exs(r)$ and the mode color.

Lemma 1.1 *For $r \geq r_k$, if $Exs(r) \geq 0$ then the color associated with $m_1(r)$ is the mode color. If $Exs(r) < 0$, then there exists a configuration of data points lying within distance r of q having color cardinalities $M(r)$, such that the color associated with $m_1(r)$ is not the mode color.*

Proof: Without loss of generality, let us assume colors are indexed by their position in $M(r)$. For $r \geq r_k$, among the k nearest neighbors at least $k - \sum_{i>1} m_i(r)$ are of color 1. Because colors are numbered in nonincreasing order, color 1 is the mode if this quantity is at least as large as the cardinality of color 2. That is, if $k - \sum_{i>1} m_i(r) \geq m_2(r)$. But it is easy to see that this is equivalent to $Exs(r) \geq 0$.

Conversely, if $Exs(r) < 0$, then consider a configuration of data points within distance r where the points of color 2 are closest to q, the points of colors 3 through c are the next closest, and points of color 1 are the furthest. Because the points of color 1 are furthest from q, there are at most $k - \sum_{i>1} m_i(r)$ points of color 1 among the k nearest neighbors of q. Because the points of color 2 are the closest to q, there are at least $\min(k, m_2(r))$ points of color 2 among the k nearest neighbors. If $m_2(r) \geq k$ then color 2 is certainly the mode color among the k nearest neighbors. Otherwise, among the k nearest neighbors, color 2 occurs at least $m_2(r) - (k - \sum_{i>1} m_i(r)) = -Exs(r)$ more often than color 1 among the k nearest neighbors. Since $-Exs(r) > 0$, this quantity is positive, implying that color 1 is not the mode. $\square$

Observe that $Exs(r)$ is a nonincreasing function of r. For $k \geq 1$, we generalize the definition of r_χ to be the largest $r \geq r_k$ such that $Exs(r) \geq 0$, that is

$$r_\chi = \sup_{r \geq r_k} \{Exs(r) \geq 0\}.$$

Assuming that points are in general position (no two points are equidistant from q) then $r_\chi > r_k$, since $m_2(r_k) \leq m_1(r_k)$ and so $Exs(r_k) \geq k - \sum_{i\geq 1} m_i(r_k) = 0$. Also observe that this generalizes the definition given for the single nearest neighbor case, since the excess is nonnegative if and only if there is only one color within distance r. For $k \geq 1$, define the *chromatic density* of a query point q to be $\delta(q) = (r_\chi - r_k)/r_k$. When q is clear from context, we will just write δ.

We believe that the chromatic density parameter intuitively captures the strength of the clustering near the query point, and hence as the parameter's value increases, query processing should be easier. Our main result for exact nearest neighbor searching establishes this relationship. Recall that a chromatic nearest neighbor query is: given a query point q and a constant $k \geq 1$, determine the mode among the k closest data points to q.

Theorem 1.1 *Let d and c be fixed constants, and let P denote a set of n data points in R^d that have been partitioned into c color classes. We can preprocess P in $O(n \log n)$ time and $O(n)$ space so that given any $q \in R^d$ and constant $k \geq 1$, chromatic k nearest neighbor queries can be answered in time $O(\log^2 n + \Delta^d \log \Delta)$, where Δ is $O(1/\delta)$.*

Thus, as long as δ is bounded away from zero, the algorithm runs in $O(\log^2 n)$ time. If δ is very close to zero, then this theorem provides no good bound on the running time of the algorithm, but in the worst case the algorithm cannot take more than $O(n \log n)$ time, the time needed to process the entire tree using our search algorithm. The constant factors in space and preprocessing time grow linearly with d and c. Constant factors in query processing grow exponentially in d, but linearly in c and k. The algorithm is relatively simple and has been implemented. Preprocessing is based on computing a balanced variant of a *k-d* tree, called a *BBD tree*, and query processing consists of a simple branch-and-bound search in this tree.

Here is an intuitive overview of the algorithm and its running time. The BBD tree defines a hierarchical subdivision of space into d-dimensional rectangles of bounded aspect ratio (that is, the ratio of the longest to shortest side). The query algorithm operates by a variant of branch-and-bound search. Each node of the BBD tree is associated with a region of space. At each stage, the algorithm maintains a set of *active nodes* among which is contained the kth nearest neighbor of the query point. Each node is associated with a *size*, which is the longest side of the corresponding rectangular region. The algorithm repeatedly *expands* the active node of the largest size, by replacing it with its two children in the tree. The algorithm computes distance estimates to the kth nearest neighbor and tests whether any existing active nodes can be made inactive. After each expansion, the algorithm computes a lower bound on the excess, and terminates as soon as this bound is nonnegative.

The $O(\log^2 n)$ term in the complexity arises from the time required by the algorithm to localize the query point in the data structure. The BBD tree has $O(\log n)$ height, and we will argue that the branch-and-bound search expands at most $O(\log n)$ nodes for each level of the tree as part of this localization. The $O(\Delta^d \log \Delta)$ term arises as follows. From properties of the BBD tree, the sizes of the regions associated with the active nodes decrease at a predictable rate. We will show that once the size of the largest active node falls below $\Omega(r_\chi - r_k)$, then the algorithm can determine the distances to the data points in the active nodes to sufficiently high precision that the mode color can be inferred. Because the active nodes correspond to disjoint regions of space of bounded aspect ratio, we will be able to apply a packing argument to show that the number of such nodes is $O(\Delta^d)$ where Δ is $O(1/\delta)$. The factor $O(\log \Delta)$ is due to the overhead of the branch-and-bound search, which maintains active nodes in a priority queue sorted by size.

If δ is close to 0, then the above analysis does not provide a very good bound on running time. For practical reasons, it is a good idea to consider earlier termination conditions in case δ is small. For this reason, we have considered the more general problem computing *approximate* nearest neighbors. 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 ϵ 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 ϵ from the true ith 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: given q, $\epsilon > 0$, and k determine the color occurring most frequently among *any* sequence of k approximate nearest neighbors. Throughout we assume that the parameters ϵ and k are supplied as a part of the query, and are not known at the time of preprocessing. Our more general result is the following.

Theorem 1.2 *Let d and c be fixed constants, and let P denote a set of n data points in R^d that have been partitioned into c color classes. We can preprocess P in $O(n \log n)$ time and $O(n)$ space so that given any $q \in R^d$, $\epsilon > 0$, and constant $k \geq 1$, approximate chromatic k nearest neighbor queries can be answered in time $O(\log^2 n + \Delta^d \log \Delta)$, where Δ is $O(1/\max(\epsilon, \delta))$.*

Observe that the worst-case running time is within a log factor of the nonchromatic approximate nearest neighbor algorithm of Arya, et al. [5] (for $k = 1$).

As mentioned earlier, the algorithm preprocesses the data into a simple data structure called a balanced box-decomposition tree (BBD tree) for query processing. This preprocessing is described in Section 2. Section 3 describes the search algorithm. Section 4 presents the changes needed to perform approximate chromatic nearest neighbor searching. Section 5 provides the query sensitive analysis of the algorithm. Finally experimental results are presented in Section 6. The experiments show (using both synthetically generated data sets and actual Landsat image data) that our algorithm outperforms the algorithm presented by Arya, et al. [5] if the data set is sufficiently well clustered.

2 Preprocessing.

The data structure we will use for answering queries is called a *balanced box-decomposition tree* (or BBD tree). The BBD tree is a balanced variant of a number of well-known data structures based on hierarchical subdivision of space into regions bounded by d-dimensional axis-aligned rectangles. Examples of this class of structures include point quadtrees [30], *k-d* trees [9], and various balanced and unbalanced structures based on a recursive subdivision of space into rectangles of bounded aspect ratio [5, 11, 12, 13, 14, 34].

The BBD tree is described in detail by Arya, et al. [6]. We present a high level overview of the data structure. For our purposes, a *rectangle* is a d-fold product of closed intervals along each of the d coordinate axes. The *size* of a rectangle is the length of its longest side. A *box* is a rectangle of bounded aspect ratio, that is, a box in which the ratio between the lengths of the longest and shortest sides are bounded by some constant. For concreteness we assume that the maximum aspect ratio is 3:1. A *cell* is either a box or the set-theoretic difference between two boxes, one nested within the other. The two boxes defining a cell are called the *inner box* and the *outer box*. Each node of a BBD tree is associated with a cell, and hence with the data points that lie within the cell. (Points lying on the boundary between two or more cells can be assigned to any one of these cells.)

The BBD tree is associated with a hierarchical subdivision of space into cells. The root is associated with the "infinite rectangle" $[-\infty, +\infty]^d$. Given an arbitrary node associated with some cell, the two children of this node are associated with a subdivision of this cell into two subcells using either of the following subdivision steps. (See Figure 2. The shaded box is the inner box of the original cell.)

Fair split: The cell is subdivided into two subcells by a hyperplane that is orthogonal to one of the coordinate axis. If the original cell has an inner box, then this box lies entirely within one of these subcells.

Shrink: The cell is subdivided into two subcells by a rectangle that is contained within the outer box and completely contains the inner box (if there is one).

Centroid Shrink: A combination of two shrinks and one split such that each of the resulting cells has at most a constant factor of the original number of data points. See [6] for details.

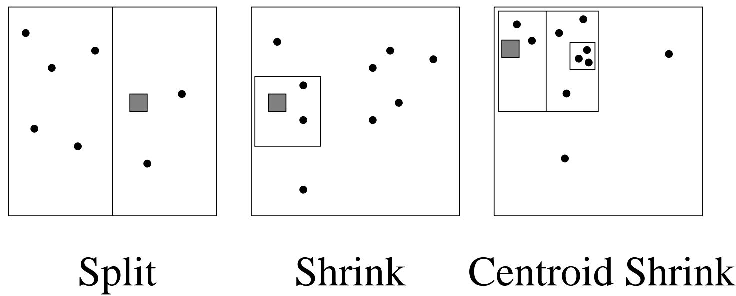

Figure 2: Balanced Box Decomposition.

There are some other technical constraints on the nature of splitting and shrinking which we do not go into here [6]. A combination of these subdivision steps are performed until each cell is associated with either zero or one data point (or more practically, a small number of data points). The resulting *leaf cells* define a subdivision of R^d into cells. It can be shown that after a split the size of a child node can be at

most a constant factor smaller than the parent node. For example, we may assume that the size of the child is at least 1/3 the size of its parent [6]. The inner child of a shrinking node can be arbitrarily smaller than its parent. We will often blur the distinction between a node, its associated cell, and its associated set of points.

In [6], it was shown that given n points in R^d, in $O(dn \log n)$ time it is possible to build a BBD tree for these points having the following properties.

Balance: The tree is of height $O(\log n)$. In general, with any 4 levels of descent, the number of points associated with a node of the tree decreases by at least a factor of 2/3.

Size reduction: There is a constant b, such that in any descent of the tree through at least b levels, the sizes of the associated cells decrease by at least 1/2. For concreteness, we may assume that $b = 4d$.

From the fact that cells have bounded aspect ratio, we have the following technical lemma, which will be important to our analysis. The proof is based on a simple density argument and is given in [6].

Lemma 2.1 (Packing Constraint) *Given a BBD tree for a set of data points in R^d the number of disjoint cells of size at least $s > 0$ that intersect any ball of radius r in any Minkowski metric is at most $\lceil 1 + 6r/s \rceil^d$.*

As the BBD tree is constructed, we assume the following additional information is associated with each node. For each node we store the total number of points associated with this node, and for each node and each color class, we store the number of points of this color associated with this node.

3 Search Algorithm.

Recall that the problem is that of determining the color occurring most frequently (the mode) among the k nearest neighbors of the query point. Recall that r_k denotes the (unknown) distance from the query point to its kth nearest neighbor, and that r_χ 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 ∞.

The algorithm applies a branch-and-bound type of search. At each stage, we maintain a set of *active* nodes. A node ceases to be active when the contribution of

its associated points can be completely determined, either because they are all so close to the query point that they are guaranteed to be among the k nearest neighbors, or else that they are so far that none of them can be among the k nearest neighbors.

The algorithm operates in a series of *node expansions*. Expanding a node means replacing an active node with its two children. Recall that the *size* of a node is the longest side length of its associated cell. The algorithm selects the largest (in size) active node for expansion[1].

The goal of the search is to minimize the number of nodes that are active at any time. For example, if it can be inferred that all the points associated with the node either lie completely outside or completely inside the ball of radius r_k, then this node need not be expanded further. For each active node v, let $dist^-(v)$ ($dist^+(v)$) denote the lower (upper) bound on the distance from the query point q to a data point stored within v. For internal nodes it is defined to be the distance from the query point to the closest (furthest) point on the boundary of the cell. This can be computed in $O(d)$ time from inner and outer boxes for the cell. For leaf nodes, the distance is determined by explicitly computing the distance from the query point to the data point associated with this leaf. If the leaf contains no data point, then both distances are set to $+\infty$. (See Figure 3(a).)

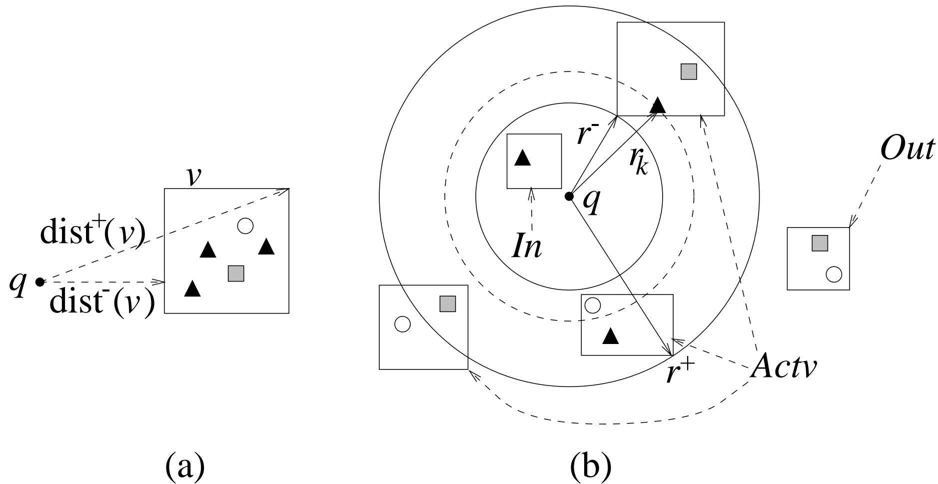

Figure 3: Elements of the search.

The algorithm also maintains three groups of nodes, *In*, *Out*, and *Actv*, such that

$$\begin{aligned} dist^+(v) \le r^- &\quad\Rightarrow\quad v \in In, \\ dist^-(v) > r^+ &\quad\Rightarrow\quad v \in Out. \end{aligned}$$

[1]The order in which cells are expanded can be based on more elaborate criteria. The rule of expanding the largest active node first is sufficient for our analysis, but other orders may lead to more rapid termination in practice. For example, alternately expanding the largest cell and closest cell to q provides an elegant way to dovetail between this algorithm and the approximate nearest neighbor algorithm of Arya, et al. [6].

Every point associated with an *In*-node lies within the ball of radius r_k, and every point associated with an *Out*-node lies outside this ball. Hence, these two subsets of nodes need not be expanded further. Nodes that satisfy neither of these conditions are placed in the group of active nodes, *Actv*. (See Figure 3(b) for an example with $k = 3$.) At all times, the union of the cells associated with these three groups of nodes forms a subdivision of space into regions with pairwise disjoint interiors, and hence induces a partition on the point set.

After expanding a node, the algorithm updates the lower and upper estimates for r_k, r^- and r^+, as follows. For each node $u \in \mathit{In} \cup \mathit{Actv}$, we assume that this node has been associated with its lower distance, $\mathit{dist}^-(u)$, and a weight equal to the number of points associated with this node. Define r^- to be the element of weight k in this weighted set of distances. (That is, the element of rank k in a multiset of distances, where each node's distance is replicated as many times as its weight.) Define r^+ similarly, but using $\mathit{dist}^+(u)$ instead. After r^+ and r^- are updated, nodes are reevaluated for "activeness", and a termination condition is tested. Termination occurs when the mode color can be determined simply from the color counts of the active cells. For example, in the single nearest neighbor case, termination occurs when all the active cells contain points of the same color.

Here is a high-level description of the algorithm. The efficient implementation of some of the steps of the algorithm will be discussed later.

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

(2) Repeat the following steps.

(a) (Expansion) Consider the largest node $v \in \mathit{Actv}$. If v is a nonleaf node, then replace v by its children v_1 and v_2 in the BBD tree. For $i = 1, 2$, compute $\mathit{dist}^-(v_i)$ and $\mathit{dist}^+(v_i)$. If v is a leaf, then replace v by the point p contained within the leaf (a cell of size 0), and set $\mathit{dist}^-(p) = \mathit{dist}^+(p)$ to be the distance from p to the query point.

(b) (Update distance estimates) Update the values of r^- and r^+ accounting for the newly expanded nodes.

(c) (Classification) For each node $u \in \mathit{Actv}$:

- if $(\mathit{dist}^+(u) \leq r^-)$ then add u to *In*.
- if $(\mathit{dist}^-(u) > r^+)$ then add u to *Out*.
- otherwise leave u in *Actv*.

(d) (Termination condition) Sum the color counts for all points in $\mathit{In} \cup \mathit{Actv}$. Let $M = \langle m_1, \ldots, m_c \rangle$ denote the nonincreasing sequence of these colors. Let $\mathit{Exs} = k - m_2 - \sum_{i>1} m_i$. If $\mathit{Exs} \geq 0$ then terminate, and return the color of m_1 as the answer.

Observe that because the algorithm always replaces one cell with a set of cells which subdivide the same region of space, the algorithm maintains the invariant that the cells associated with the current set of nodes subdivide space, and hence the points have been implicitly partitioned into the sets *In*, *Actv*, and *Out*. Also observe that because $dist^{-}(v)$ is a lower bound on the distance from the query point to each of the points stored in node v, it follows that r^{-} (r^{+}) is a lower (upper) bound on r_k. Thus, points classified as being in *In* (*Out*) are indeed closer (further) to the query point than the kth nearest neighbor, as desired.

The following lemma, combined with the fact that the set $In \cup Actv$ contains the k-nearest neighbors, implies that the algorithm returns a correct answer. The proof is essentially the same as the first half of the proof of Lemma 1.1.

Lemma 3.1 *Let V denote any set of nodes in the BBD tree whose associated cells are pairwise disjoint, and which contains the k nearest neighbors of the query point. Let $M(V) = \langle m_1(V), \ldots, m_c(V) \rangle$ denote the nonincreasing sequence of color counts among all points in the associated cells. Let $Exs = k - m_2(V) - \sum_{i>1} m_i(V)$. If $Exs \geq 0$, then the color associated with $m_1(V)$ is the mode among the k-nearest neighbors.*

Before analyzing the algorithm's running time, we describe some elements of the algorithm's implementation. The first issue involves the selection of the active node to be expanded at each iteration. The simplest way to handle this efficiently is to store all the current active nodes in a priority queue according to node size. The time to select each node for expansion would then be proportional to the logarithm of the number of active nodes. We can improve upon this slightly (ultimately saving a factor of $O(\log \log n)$ in our analysis) by observing that if a cell c is nested within the outer box of another cell c', then the node associated with c cannot be a candidate for the largest expanded node, because c' must be at least as large. Thus, for the purpose of selecting the node for expansion, the priority queue need contain only the *maximal* nodes whose associated cells are not nested within the outer box of any other active cell.

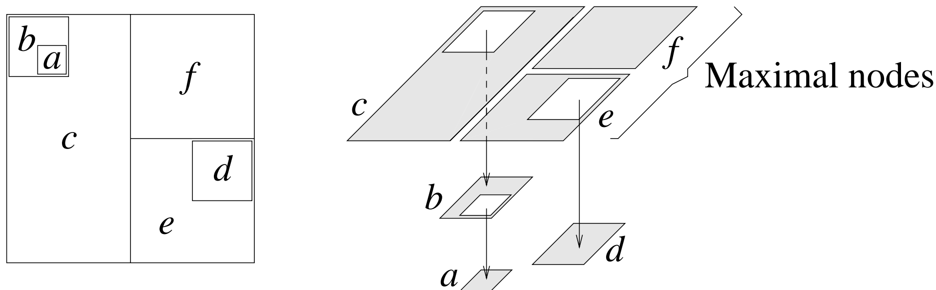

Figure 4: Maximal nodes.

We handle this by maintaining a linked list for each of the maximal nodes, where each node contains a pointer to the active cell nested immediately within it. Each

nested node points to the active cell nested immediately within it, and so on. (See Figure 4.) Since each cell has exactly one inner box, these nested cells form a simple linear list. It is not hard to show that the length of each of these chains is no greater than the height of the BBD tree, $O(\log n)$. Observe that when some maximal node is classified as being in *Out*, because its closest distance to the query point exceeds r^+, it follows that the closest point of all nested nodes exceeds r^+, and hence all can be assigned to *Out* at once. Similarly, if a maximal node is classified as being in *In*, because its furthest distance to the query point is less than r^-, all nested nodes are closer than r^-, and hence all can be assigned to *In* at once. Whenever we attempt to expand an active leaf node whose associated cell contains an inner box, we "awaken" the largest active node that is nested within the leaf's cell. This is done by making adding the node at the head of the linked list to the set of maximal active nodes, and the remainder of the linked list is then associated with this new maximal node.

Although it is theoretically possible that each maximal active node could have as many as $O(\log n)$ nodes nested within it, our experience with data sets from a variety of distributions suggests that shrinking is so rarely needed (indeed, it was never needed in any of our experiments) that the nested linked lists can be avoided to simplify the implementation without significantly affecting performance.

Once it has been determined which node will be expanded, the actual expansion can be performed in constant time, and the distances to this node can be computed in $O(d)$ time. Next we consider how r^+ and r^- are updated. This could be done by invoking a weighted selection algorithm on the set of active nodes. However, the running time of the selection will be linear in the number of active nodes. Since we are only interested in the kth nearest neighbor, where k is a constant, there is a more efficient way.

The nodes of $In \cup Actv$ are maintained in two data structures, one sorted by $dist^-$ and the other by $dist^+$. We will describe the data structure for $dist^-$ first. We store these nodes in a combined data structure. We maintain a *hotlist* of nodes for which $dist^- \leq r^-$ in a linked list, and store the remaining nodes of $In \cup Actv$ in a fast *priority queue* (e.g., a binary heap [18]). When a node is expanded it is replaced by two nodes with different weights and different distances. The original node is removed from data structure, and its two children are inserted into the priority queue. If the original node came from the hotlist, then we repeatedly extract the minimum entry from the priority queue until the total weight of nodes in the hotlist is at least k. The value of r^- is updated by considering the maximum value of $dist^-$ for the nodes in the hotlist. The case of $dist^+$ is similar. In fact it is actually simpler, since it can be shown that the priority queue for $dist^+$ is not needed. A node cannot enter the hotlist unless one its ancestors is already in the hotlist.

Letting A denote the number of maximal active nodes, we claim that in $O(k + \log A)$ amortized time it is possible to perform this part of the expansion. Because each node contains at least one point, there can be at most k nodes in the hotlist at any time. Deleting and inserting a constant number of nodes into the hotlist,

and updating r^- and r^+ can be performed in $O(k)$ time. Inserting the two children of the expanded node into the priority queue can be performed in $O(\log A)$ time. Each extraction from the priority queue can be performed in $O(\log A)$ time, but if we amortize this time to extract each node against the $O(\log A)$ time needed to insert the node, then we may ignore this component of the cost.

The final issue to consider is the evaluation of the termination condition. This can be done by maintaining the color counts m_i, $1 \leq i \leq c$, for the active nodes with each expansion. When a node is killed, we subtract its color counts from this list of color counts for the active nodes. This can be done easily in $O(c)$ time per expansion. In the same time we can test the termination condition (since the largest and second largest color counts can be found in $O(c)$ time).

Combining these observations, we have the following result.

Lemma 3.2 *Each expansion can be processed in* $O(k + c + \log A)$ *amortized time, where* A *denotes the number of maximal active nodes,* k *is the number of nearest neighbors, and* c *is the number of color classes. Under our assumption that* k *and* c *are constants, this is* $O(\log A)$*.*

4 Approximate Nearest Neighbors

It is an easy matter to generalize the algorithm presented in the previous section to compute the mode color among some sequence of k approximate nearest neighbors (recall the definition in the Introduction). In particular, we add the following additional termination condition just after step 2(d).

(2)

$\vdots$

(e) (Termination condition 2) Let s denote the size of the largest node in *Actv*. If $s \leq \epsilon r^-/d$ then enumerate a set of k witnesses to r^-, and return the mode color of this set.

By a set of *witnesses* to r^-, we mean a set of k data points formed by taking all the data points in nodes v in $In \cup Actv$ for which $dist^+(v) < r^-$ (there can be at most k such points by definition of r^-) and filling out the set with sufficiently many points chosen from any nodes v in $In \cup Actv$ for which $dist^+(v) = r^-$ (there must be at least k total points by definition of r^-). We show below that this forms a set of k approximate nearest neighbors to q. Note that we can bound the number of nodes in *Actv*, but we cannot bound the number of data points in all of these nodes. Thus it is not possible to enumerate these points in exact order of distance from q.

If this termination condition is satisfied, then the following lemma establishes the correctness of the approximation algorithm.

Lemma 4.1 *If Termination condition 2 is satisfied, then the k witnesses to r^- that are closest to the query point, are a sequence of k approximate nearest neighbors to q.*

Proof: Consider the k closest witnesses. If one of these points was associated with a node v in *In*, then $dist^+(v) \leq r^- \leq r_k$, implying that the point is among the k-nearest neighbors. On the other hand, if the point was associated with a node v in *Actv*, then $dist^-(v) \leq r^-$. This node is associated with a cell of size at most s, where $s \leq \epsilon r^-/d$, from the termination condition. In any Minkowski metric, the diameter of a cell whose maximum side length is s is ds (the worst case arising in the L_1 metric), and hence

$$dist^+(v) \ \leq \ dist^-(v) + ds \ \leq \ r^- + \epsilon r^- \ \leq \ (1+\epsilon)r^- \ \leq \ (1+\epsilon)r_k.$$

Thus each of these witnesses is close enough to be an approximate kth nearest neighbor to the query point. Finally, because these points were selected from the closest nodes to the query point, it follows that there can be no significantly closer data points that were not included. Therefore, the sorted sequence of witnesses is a sequence of k approximate nearest neighbors, and the mode color of this set is the desired color.
□

5 Query Sensitive Analysis.

In this section we present the query sensitive analysis stated in Theorem 1.2 by showing that the query algorithm terminates in $O(\log^2 n + \Delta^d \log \Delta)$ time, where n is the number of data points, Δ is $O(1/\max(\epsilon, \delta))$, and where ϵ is the user-supplied approximation bound and δ is the chromatic density of the query. Recall that we assume that the dimension d, number of colors c, and number of nearest neighbors k are constants.

The proof is based on an analysis of the number of active nodes expanded by the algorithm. It is similar in structure to the proof of the number of nodes expanded in the approximate range query algorithm by Arya and Mount [7]. In our case, the analysis is complicated by the fact that the radii r^+ and r^- vary dynamically throughout the algorithm.

A node v is said to be *visited* if the node is ever active, and it is *expanded* if its children are visited. We distinguish between two kinds of expanded nodes depending on size. An expanded node v for which $size(v) \geq r_\chi$ is *large* and otherwise it is *small*. Because the sizes of the expanded nodes decrease monotonically, the algorithm begins by expanding some number of large nodes, and then it switches to expanding small nodes. We refer to these as the *large phase* and *small phase*, respectively. We will show that the number of nodes expanded during the large phase is bounded by $O(\log^2 n)$. The number of maximal nodes active during each step of the large phase is $O(1)$ (with constant factors depending on dimension). Then we will show that the number

of nodes expanded during the small phase is Δ^d where Δ is $O(1/\max(\epsilon, \delta))$. This quantity also bounds the number of nodes active at any time in this phase. From Lemma 3.2 we know that each expansion takes $O(\log \Delta)$ time, and hence the running time of the small node phase is $O(\Delta^d \log \Delta)$. The total running time of the algorithm is the sum of these two components.

In the analysis below we have included the constant factors that depend on dimension for completeness. These factors are crude upper bounds, and we have made no attempt to minimize them. In Section 6 we will see that the actual constant factors appear to be quite reasonable in practice. Before presenting the analysis of the various phases, we present the following technical lemma, which establishes bounds on the radius estimates as a function of the size of the current node.

Lemma 5.1 *Let s denote the size of the most recently expanded node. Then*

$$\begin{aligned} r^- &\geq r_k - ds \qquad \textit{and} \\ r^+ &\leq r_k + ds. \end{aligned}$$

Proof: Since the largest active node is expanded first, every active node is of size at most s. In any Minkowski metric, a rectangle whose longest side length is s has diameter at most ds, and thus each active node is of diameter at most ds. Let S^+ denote the union of In and the subset of active nodes v such that $dist^-(v) \leq r_k$. Because no node in Out can intersect the ball of radius r_k centered at the query point, it follows that these nodes disjointly cover this ball, and hence their total weight is at least k. Furthermore, because each node of S^+ corresponds to a cell that is either entirely contained within this ball (if it is in In) or is at least partially contained within the ball and has diameter at most ds (if it is active), it follows that the value of $dist^+(v)$ for any node in S^+ is at most $r_k + ds$. Thus, by considering just these nodes, it follows that among $In \cup Actv$ there is a total weight of at least k among a set of nodes for which $dist^+$ is at most $r_k + ds$. Thus, by definition, $r^+ \leq r_k + ds$.

To establish the bound on r^-, let S^- denote the union of In and the subset of active nodes v such that $dist^+(v) < r_k$. These nodes are properly contained within the ball of radius r_k centered at the query point, and so do not contain all k nearest neighbors of q. Since the corresponding cells are mutually disjoint, their total weight is strictly less than k, implying that r^- is not defined by the $dist^-$ value of any of these nodes. Thus, the node $v \in In \cup Actv$ that defines r^- satisfies $dist^+(v) \geq r_k$. Such a node must be active (since it cannot be in In) and hence is of diameter at most ds. This implies that $dist^-(v) \geq r_k - ds$. Therefore, $r^- \geq r_k - ds$. □

5.1 Large-node Phase

Intuitively, the large-node phase can be viewed as the phase in which the algorithm is attempting to localize the search for the ball of radius r_χ. We will show that when expanding a large node v of size s, its distance from the query point, $dist^-(v)$, will be

proportional to s. From the packing constraint, Lemma 2.1, it will follow that only a constant number (depending on dimension) of nodes proportional to this size can be expanded. But because we have no bound on the size of r_χ (other than the precision of the arithmetic representation), it is difficult to bound the number of different sizes that we may encounter in the search. What saves us is the fact that the height of the tree is $O(\log n)$. We will argue that only $O(\log^2 n)$ different size groups can be visited. The extra log factor seems to be an unavoidable consequence of branch-and-bound search, since there is an example that requires this much time even to find the single nearest neighbor in a nonchromatic setting using the branch-and-bound paradigm [3].

To begin the analysis, we say that a node is *significant* if it is a large active node that overlaps the ball of radius r_k centered at the query point. Observe that if a node is significant, then its parent is significant, and hence all of its ancestors are significant. We begin by bounding the number of significant nodes.

Lemma 5.2 *There are at most $O(7^d \log n) = O(\log n)$ significant nodes.*

Proof: Consider the *minimal* significant nodes, that is, those neither of whose children is significant. (Note that there is no relationship between the notions of minimal introduced here and maximal introduced in Section 3.) Clearly the significant nodes are just the (nonproper) ancestors of these nodes. The cells associated with two minimal significant nodes are disjoint because neither node can be an ancestor of the other in the tree. Each minimal significant node is of size at least $r_\chi \geq r_k$, and each overlaps a ball of radius r_k centered at the query point. Thus, by applying the packing constraint, Lemma 2.1, it follows that there are at most $\lceil 1 + 6r_k/r_k \rceil^d = 7^d$ minimal significant nodes. By including all the ancestors of these nodes, there are at most $O(7^d \log n)$ significant nodes. □

At any stage of the algorithm, define the *dominant node* to be the significant node of largest size. It is possible that the last large significant node to be expanded by the algorithm results in an active node which overlaps the ball of radius r_k but whose size is less than r_χ. For the sake of the argument below, let us think of this one "insignificant" node as being significant. Since throughout this large phase there is always at least one active significant node (e.g., the one that contains the kth closest data point), there is always a dominant node. Since dominant nodes are significant, Lemma 5.2 provides an upper bound on the number of dominant nodes.

The analysis of the large phase rests on showing that while any one node is dominant, we can use the packing constraint to bound the number of nodes that are expanded.

Lemma 5.3 *Let v be a dominant node. While v is dominant:*

(i) at most $O((6d + 7)^d \log n) = O(\log n)$ nodes can be expanded by the algorithm and,

(ii) the number of maximal active nodes during any one expansion is at most $(18d+19)^d = O(1)$.

Proof: Let s denote the size of v. Since v is dominant, and hence significant, we know that $s \geq r_\chi \geq r_k$. Any node that partially overlaps the ball of radius r_k centered at the query point is active, and hence is of size less than s (either because the node is small, or because the node is large, hence significant, and hence no larger than v). From Lemma 5.1 we have $r^+ \leq r_k + ds \leq (d+1)s$.

Any node w that is expanded while v is dominant must overlap the ball of radius r^+ centered at the query point (or else it would not be active), and must be at least as large as v (for otherwise, v would already have been expanded). Consider a fixed level of the BBD tree, and consider the set of nodes on this level that are expanded while v is active. The cells associated with the nodes of this set are pairwise disjoint (since overlap only occurs between ancestors and descendents), they are of size at least s, and they overlap a ball of radius $(d+1)s$. By the packing constraint, there are at most $\lceil 1 + 6(d+1)s/s \rceil^d = (6d+7)^d$ such cells. Summing over all levels of the tree gives $O((6d+7)^d \log n)$ nodes expanded while v is dominant. This establishes (i).

To prove (ii), consider the set of maximal active nodes during some expansion while v is active. We claim that we can identify this set with a set of disjoint cells of size at least $s/3$ that overlap the ball of radius r^+. In particular, consider a maximal active node w. If w arose from a splitting operation, it is the child of a node of size at least s (because the parent must have been expanded before v) and hence (from the comments made in Section 2) is of size at least $s/3$. If w came about as an outer box in shrinking, it is of the same size as its parent, which is at least s. If it came about as an inner box in shrinking, the outer cell cannot be active as well (for otherwise w would not be maximal), and therefore we can identify this node with its union with the outer cell. Thus, the cells of the resulting set are of size at least $s/3$, they are disjoint, and because they are all active, each overlaps the ball of radius $r^+ \leq (d+1)s$ centered at the query point. Therefore, by applying the packing constraint, it follows that the number of maximal active cells is at most $\lceil 1 + 6(d+1)s/(s/3) \rceil^d = (18d+19)^d = O(1)$. □

Combining Lemmas 5.2 and 5.3(i) and using the fact that all dominant nodes are significant, it follows that the total number of expanded nodes during the large phase is the product of these two bounds, yielding a total of $O((7(6d+7))^d \log^2 n) = O(\log^2 n)$ nodes expanded during the large-node phase. From Lemma 5.3(ii), it follows that the number of maximal active nodes is $O((18d+19)^d) = O(1)$.

5.2 Small-node Phase.

Next we consider the case in which nodes are of size less than r_χ. In this phase, the algorithm is attempting to ascertain the distance between the query point and data points of the active cells to sufficiently high precision that it can satisfy either

of the algorithm's two termination conditions. The analysis is based on determining a crude lower bound on the size of the smallest node to be expanded by this phase of the search, and a crude upper bound on the distance of any such node from the query point. Using the packing constraint, we can bound the number of disjoint nodes of this size that lie within this distance of the query point. One important property of the BBD-tree is that with each constant number of levels of descent through the tree, the size of the nodes decreases by a constant factor. The final bound comes by summing over the various cell sizes encountered in the search.

Our first observation bounds the size of nodes at which termination is guaranteed.

Lemma 5.4 *For all sufficiently small ϵ, when the size of the currently expanded node is less than*

$$s_{\min} = \max(\epsilon, \delta)\frac{r_k}{2d},$$

then the algorithm terminates.

Proof: Let us assume that $\epsilon < 1$. (Any constant can be used, but the constant factors derived below will be affected.) Let $s < s_{\min}$ denote the size of the currently expanded node. From Lemma 5.1 we have

$$\begin{aligned} r^- &\geq r_k - ds_{\min} = r_k - \max(\epsilon, \delta)\frac{r_k}{2} = r_k\left(1 - \frac{\max(\epsilon, \delta)}{2}\right), \\ r^+ &\leq r_k + ds_{\min} = r_k + \max(\epsilon, \delta)\frac{r_k}{2} = r_k\left(1 + \frac{\max(\epsilon, \delta)}{2}\right). \end{aligned}$$

We consider two cases. First, if $\epsilon \geq \delta$, then from our assumption that $\epsilon < 1$, we have $r^- \geq r_k(1 - \epsilon/2) \geq r_k/2$. Thus, we have $s \leq \epsilon r_k/(2d) \leq \epsilon r^-/d$. Therefore, the algorithm terminates by Termination condition 2.

On the other hand, if $\delta > \epsilon$, we have

$$r^+ \leq r_k\left(1 + \frac{\delta}{2}\right) = r_k\left(1 + \frac{r_\chi - r_k}{2r_k}\right) = \frac{r_\chi + r_k}{2}.$$

Since each active node must overlap a ball of radius r^+ centered at the query point, and since each is of diameter at most $sd \leq \delta r_k/2 = (r_\chi - r_k)/2$, it follows that for every active node v we have

$$dist^+(v) \leq r^+ + sd \leq \frac{r_\chi + r_k}{2} + \frac{r_\chi - r_k}{2} = r_\chi.$$

In other words, all the active cells lie within the ball of radius r_χ.

Let $\langle m'_1, \ldots, m'_c\rangle$ denote the nonincreasing sequence of colors in these active cells, and if we let $\langle m_1, \ldots, m_c\rangle$ denote the colors of the points within distance r_χ of the query point, we have $m'_i \leq m_i$, for $1 \leq i \leq c$. Thus, by definition of r_χ,

$$Exs = k - m'_2 - \sum_{i>1} m'_i \geq k - m_2 - \sum_{i>1} m_i \geq 0,$$

and the algorithm terminates by the termination condition of step 2(d). □

Thus, the size of each expanded node in the small-node phase lies in the half-open interval $[s_{\min}, r_\chi)$. The rest of the analysis of the small-node phase is based on subdividing this interval into a set of *size groups*, analyzing the number of nodes expanded within each size group, and then summing over all size groups. For $i \geq 0$, define the ith size group to be the set of expanded nodes whose sizes lie within the interval $r_\chi[1/2^{i+1}, 1/2^i)$. Clearly the last size group to be expanded is group $i_{\max} = \lfloor \lg(r_\chi/s_{\min}) \rfloor$.

To determine the number of expanded nodes in a given size group, we will relate them to a set of disjoint cells that overlap a ball of a given radius, and then apply the packing constraint. We know that a cell is active, and hence expanded only if it overlaps the ball of radius r^+ centered at the query point. First we show that the algorithm's distance estimate r^+ can be related to r_χ. From Lemma 5.1 and the fact that $r_k \leq r_\chi$ we have the following.

Lemma 5.5 *Throughout the small-node phase,* $r^+ \leq (d+1)r_\chi$.

The problem with cells of a given size group is that they will generally not be disjoint. However, we can identify a subset of disjoint nodes and argue that the cardinality of this subset is within a constant factor of the set we wish to bound. Intuitively the reason is that with a constant number of levels of descent in the BBD tree, the sizes of the nodes decrease by a constant fraction. Thus, we cannot have arbitrarily long chains of overlapping nodes within a given size group. For $i \geq 0$, consider the nodes forming the ith size group. For each pair of nodes u and v, such that u is an ancestor of v in the BBD tree (and hence, u and v's cells overlap one another) remove u from the set. The set resulting by repeating this operation for all nodes in the size group is called the ith *reduced size group.*

Lemma 5.6 *For* $i \geq 0$*, the number of nodes in the* i*th size group is at most* $4d$ *times the number of nodes in the* i*th reduced size group.*

Proof: Recall that two nodes have overlapping cells if and only if one is an ancestor of the other in the BBD tree. Also recall from Section 2 that with every $4d$ levels of descent in the tree, the sizes of the nodes decrease by at least half, and hence belong to a different size group. Thus, if a node v is in the ith size group, it could have resulted in the elimination of at most $4d$ of its ancestors in the same size group. Thus the size group has at most a factor of $4d$ additional nodes. □

We now apply a packing argument to each reduced size group to bound the number of nodes expanded in the corresponding size group. This is the main result for the small-node phase analysis.

Lemma 5.7 *The total number of nodes expanded during the small-node phase is at most*

$$16d^2 \left(\frac{1 + 6(d+1)}{\max(\epsilon, \delta)} \right)^d = O(\Delta^d),$$

where Δ is $O(1/\max(\epsilon, \delta))$. This also bounds the number of nodes active at any time in this phase.

Proof: For $i \geq 0$, let N_i denote the number of active nodes expanded in the ith size group. First consider the ith reduced size group. The nodes of this group are disjoint, of size at least $r_\chi/2^{i+1}$, and they overlap a ball of radius r^+. From Lemma 5.5 we have $r^+ \leq (d+1)r_\chi$. From Lemma 2.1 the number of such cells is at most

$$\left\lceil 1 + \frac{6(d+1)r_\chi 2^{i+1}}{r_\chi} \right\rceil^d = (1 + 6(d+1)2^{i+1})^d.$$

From Lemma 5.6 there are at most an additional factor of $4d$ nodes in the size group, from which it follows that

$$N_i \;\leq\; 4d(1 + 6(d+1)2^{i+1})^d \;\leq\; 4d(1 + 6(d+1))^d (2^d)^{i+1}.$$

The total number of nodes expanded during the small-node phase is equal to $\sum_i N_i$, where i ranges over all the size groups. Ignoring the constant factor of $a = 4d2^d(1 + 6(d+1))^d$ this is a sum of the form $\sum (2^d)^i$, which is a geometric series. The largest term of the sum is the term generated by the smallest size group $i_{\max} = \lfloor \lg r_\chi / s_{\min} \rfloor$, which evaluates to

$$N_{i_{\max}} \leq a(2^d)^{i_{\max}} \leq a \left(\frac{r_\chi}{s_{\min}} \right)^d.$$

Because the base of the geometric series is larger than 2, it follows from a simple inductive argument that the sum is bounded by twice this largest term. Substituting $s_{\min} = \max(\epsilon, \delta) r_k / (2d)$ it follows that the total number of nodes expanded during the small-node phase is at most

$$2a(2d)^d \left(\frac{r_\chi}{\max(\epsilon, \delta) r_k} \right)^d.$$

At this point we consider two cases. First, if $r_\chi \leq 2r_k$, then this expression is at most $2a(4d/\max(\epsilon, \delta))^d$. Otherwise, if $r_\chi > 2r_k$ then we have

$$\frac{r_\chi}{\max(\epsilon, \delta) r_k} \;\leq\; \frac{r_\chi}{\delta r_k} \;=\; \frac{r_\chi}{r_\chi - r_k} \;\leq\; \frac{r_\chi}{r_\chi - (r_\chi/2)} \;\leq\; 2.$$

Thus, the expression above is at most $2a(4d)^d$. Combining the two cases, the total number of nodes expanded in the small-node phase is

$$2a(4d)^d \max\left(1, \frac{1}{\max(\epsilon, \delta)^d} \right) \;=\; O(\Delta^d),$$

where $\Delta = O(1/\max(\epsilon, \delta))$. □

Finally, we can derive the total complexity of the algorithm. From Lemma 5.3 it follows that the total number of nodes expanded during the large phase is $O(\log^2 n)$, and that the number of nodes active at any time in this phase is $O(1)$. Thus from Lemma 3.2 the total execution time for this phase, the number of nodes times the logarithm of the number of active nodes is $O(\log^2 n)$. From Lemma 5.7 above, we have the number of nodes expanded during the small phase is $O(\Delta^d)$, and since this also bounds the number of nodes active at any time in this phase, the total running time of the small-node phase is $O(\Delta^d \log \Delta)$. Combining these completes the proof of Theorem 1.2.

6 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. [5] (called *ann*). Both algorithms use the BBD tree (although the tree of [5] contains no chromatic information).

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. (Because partial distance calculation was used, not all coordinates of each visited point need be accessed.) In the plots shown below, we present only a representative subset of the data which was gathered. For both algorithms, we validated the results of each nearest neighbor calculation off-line, and computed the *effective epsilon*, that is, the actual relative error of each query. As observed in [5], average effective errors were usually quite a bit smaller than the user supplied ϵ (almost always less than 0.1 and often less than 0.01 times this value). For this reason, our experiments were often run with even relatively large values of ϵ (ranging from 0 to 10 in our experiments).

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.

The first set of experiments consisted of data sets in dimension 8 generated synthetically by a pseudo-random number generator. The distribution is called *clusnorm*. For each distribution, 10,000 data points were generated as follows. 10 cluster centers were chosen uniformly from an 8-dimensional unit hypercube, and each data

point was generated by a Gaussian variable with a given standard deviation centered around a randomly chosen center. Each cluster center was chosen with equal probability. Query points were generated using the same distribution (with the same cluster centers), and statistics were averaged over 300 queries.

As the standard deviation decreases, the distribution exhibits a higher degree of clustering, and so we expect better performance from cann. In Figure 5 we present the results of this experiment for 3 selected values of ϵ. The x-axis indicates the standard deviation of the distribution, and the y-axis indicates the average number of nodes visited for each query. Solid lines show runs of the cann algorithm and dashed lines show runs of the ann algorithm. As expected, ann is relatively insensitive to clustering, and cann performs better than ann for highly clustered data sets (small cluster standard deviations).

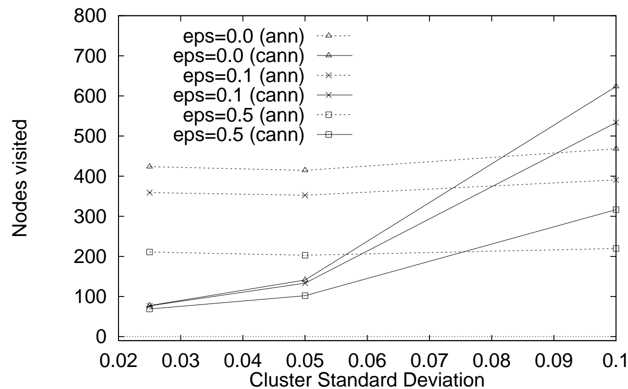

Figure 5: Experiments with synthetically generated data.

It is worth noting that the data must be very well clustered before these improvements are evident. This seems to be a consequence of the fact that the cann operates in an inherently top-down manner in its processing of the box-decomposition tree whereas the ann algorithm operates in a bottom-up manner. It seems that bottom-up processing allows ann to establish better bounds on the distance to the nearest neighbor at the early stages of the algorithm, but it is not clear how this observation can be applied to improve the performance of cann.

For our second experiment, we tested the performance of the cann algorithm on data sets arising from actual satellite images of the earth. 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 the first data set all 7 classes participated, with the following frequencies.

Color	Class	Freq.	Percent
1	Urban	510	5.10%
2	Agriculture	1670	16.70%
3	Rangeland	1539	15.39%
4	Forestland	313	3.13%
5	Water	3223	32.23%
6	Wetland	32	0.32%
7	Barren	2713	27.13%

Analysis of the spectral signatures of these classes revealed that some classes (such as water) were highly clustered and well separated from the others, but others (such as urban and rangeland) were neither well clustered nor well separated. For this reason, a second, more highly clustered and separated data set consisting of 10,000 entries was sampled with the following frequencies.

Color	Class	Freq.	Percent
4	Forestland	1000	10.00%
5	Water	4500	45.00%
7	Barren	4500	45.00%

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 the data points), and running each against one of the 10,000-entry data files. We only tested query points whose classes appeared in the data set. Experiments were run with ϵ values ranging from 0.1 to 10.

In Figure 6 and 7 we show the results of three 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 ϵ. As before, solid lines represent the cann algorithm and dashed lines represent the ann algorithm. 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.

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.

7 Conclusions

We have presented an algorithm for chromatic nearest neighbor searching, and a query sensitive analysis of its running time. This analysis rigorously establishes the

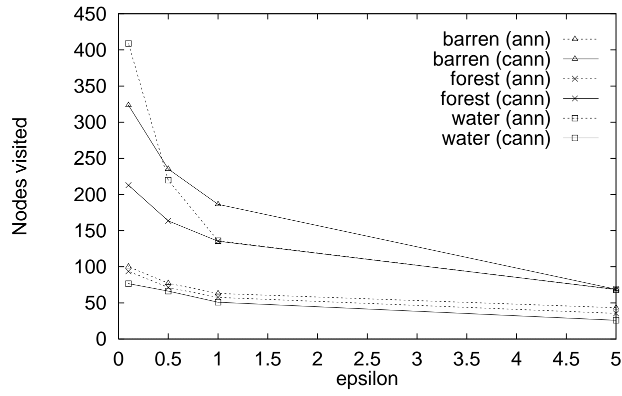

Figure 6: Landsat experiment 1.

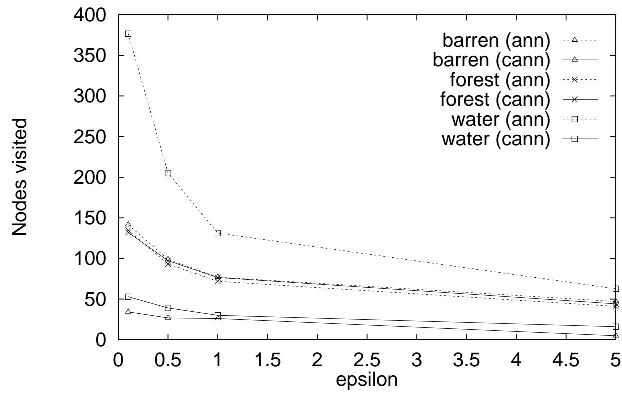

Figure 7: Landsat experiment 2.

intuitive observation that chromatic nearest neighbor searching can be performed most efficiently when color classes form well-separated clusters.

It is not difficult to extend our results to a query sensitive analysis of the nonchromatic nearest neighbor problem as well. In particular, if we imagine each point to have its own individual color, it is not hard to show that the running time of an exact nearest neighbor problem is $O(\log^2 n + \Delta^d)$ where Δ is $O(r_1/(r_2 - r_1))$, and where r_i is the distance from the query point to its ith nearest neighbor. In other words, if there is a significant relative difference in distance between the nearest and second nearest neighbors, an approach based on BBD tree search runs in polylogarithmic time.

We conjecture that the chromatic nearest neighbor algorithm can be improved by a factor of $\log \Delta$. This factor was needed to process the priority queue containing the active nodes in the small-node phase. However, in this phase the node sizes are assumed to be changing so gradually that the priority queue is not really necessary. (It is needed for the large-node phase.) Unfortunately, the algorithm does not know which phase it is operating in, since phases are defined in terms of the unknown value of r_χ.

Given that the approximate chromatic nearest neighbor is a strictly weaker problem (in an information theoretic sense) than the approximate nearest neighbor problem, it is surprising that we cannot seem to match the running time of the algorithm of Arya, et al. [5] for poorly clustered data sets, either asymptotically or experimentally. This seems to be a consequence of the branch-and-bound paradigm and the added overhead it incurs. It is an interesting problem to create a combined approach that performs well for both well- and poorly-clustered data sets.

Acknowledgements

The authors would like to thank Simon Kasif and Alon Efrat for suggesting the chromatic version of the nearest neighbor problem. We would also like to thank Sunil Arya for his helpful insights on the analysis of the branch-and-bound search.

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] H. C. Andrews. *Introduction to Mathematical Techniques in Pattern Recognition*, Wiley, New York, NY, 1972.

[3] S. Arya. Private communication, 1995.

[4] 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.

[5] 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.

[6] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *J. ACM*, 45:891–923, 1998.

[7] S. Arya, D. M. Mount. Approximate Range Searching. *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, 172–181.

[8] 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.

[9] J. L. Bentley. K-d trees for semidynamic point sets. In *Proc. 6th Ann. ACM Sympos. Comput. Geom.*, pages 187–197, 1990.

[10] M. Bern. Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45:95–99, 1993.

[11] S. N. Bespamyatnikh. An optimal algorithm for closest pair maintenance. *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, 152–161.

[12] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional pointsets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42:67–90, 1995.

[13] P. B. Callahan and S. R. Kosaraju. Algorithms for dynamic closest pair and n-body potential fields. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, 1995.

[14] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Ann. IEEE Sympos. on the Found. Comput. Sci.*, pages 226–232, 1983.

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

[16] K. L Clarkson. An Algorithm for Approximate Closest-Point Queries. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pages 160–164.

[17] J. G. Cleary. Analysis of an algorithm for finding nearest neighbors in Euclidean space. *ACM Transactions on Mathematical Software*, 5(2):183–192, 1979.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1990.

[19] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, NY, 1973.

[20] K. Fukunaga and P. M. Narendra. A branch-and-bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.*, 24:750–753, 1975.

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

[22] 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.

[23] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, Englewood Cliffs, NJ, 1986.

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

[25] Q. Jiang and W. Zhang. An improved method for finding nearest neighbors. *Pattern Recognition Letters*, 14:531–535, 1993.

[26] B. Kamgar-Parsi and L. N. Kanal. An improved branch-and-bound algorithm for computing k-nearest neighbors. *Pattern Recognition Letters*, 3:7–12, 1985.

[27] H. Niemann and R. Goppert. An efficient branch-and-bound nearest neighbour classifier. *Pattern Recognition Letters*, 7:67–72, 1988.

[28] R. L. Rivest. On the optimality of Elias's algorithm for performing best-match searches. In *Information Processing*, pages 678–681. North Holland Publishing Company, 1974.

[29] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[30] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley, Reading, MA, 1990.

[31] M. R. Soleymani and S. D. Morgera. An efficient nearest neighbor search method. *IEEE Transactions on Communications*, 35(6):677–679, 1987.

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

[33] J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles*. Addison-Wesley, Reading, MA, 1974.

[34] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

[35] A. C. Yao and F. F. Yao. A general approach to d-dimensional geometric queries. In *Proc. 17th Ann. ACM Sympos. Theory Comput.*, pages 163–168, 1985.