Chromatic k-Nearest Neighbor Queries

Thijs van der Horst
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract

Let P be a set of n colored points. We develop efficient data structures that store P and can answer chromatic k-nearest neighbor (k-NN) queries. Such a query consists of a query point q and a number k, and asks for the color that appears most frequently among the k points in P closest to q. Answering such queries efficiently is the key to obtain fast k-NN classifiers. Our main aim is to obtain query times that are independent of k while using near-linear space.

We show that this is possible using a combination of two data structures. The first data structure allow us to compute a region containing exactly the k-nearest neighbors of a query point q, and the second data structure can then report the most frequent color in such a region. This leads to linear space data structures with query times of $O(n^{1/2}\log n)$ for points in $\mathbb{R}^1$, and with query times varying between $O(n^{2/3}\log^{2/3} n)$ and $O(n^{5/6}\operatorname{polylog} n)$, depending on the distance measure used, for points in $\mathbb{R}^2$. These results can be extended to work in higher dimensions as well. Since the query times are still fairly large we also consider approximations. If we are allowed to report a color that appears at least $(1-\varepsilon)f^*$ times, where f^* is the frequency of the most frequent color, we obtain a query time of $O(\log n + \log\log_{\frac{1}{1-\varepsilon}} n)$ in $\mathbb{R}^1$ and expected query times ranging between $\tilde{O}(n^{1/2}\varepsilon^{-3/2})$ and $\tilde{O}(n^{1/2}\varepsilon^{-5/2})$ in $\mathbb{R}^2$ using near-linear space (ignoring polylogarithmic factors).

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases data structure, nearest neighbor, classification

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.67

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2205.00277`

Funding *Maarten Löffler*: Partially supported by the Dutch Research Council (NWO) under the project numbers 614.001.504 and 628.011.005.

Acknowledgements We would like to thank an anonymous reviewer for the randomized solution presented in Section 3.2.1, which led to our current solution for finding $\mathcal{D}_2^k(q)$ in Section 3.2.2.

1 Introduction

One of the most popular approaches for classification problems is to use a k-Nearest Neighbor (k-NN) classifier [3, 10, 12, 14]. In a k-NN classifier the predicted class of a query item q is taken to be the most frequently appearing class among the k items most similar to q. One can model this as a geometric problem in which the input items are represented by a set P of n colored points in $\mathbb{R}^d$: the color of the points represents their class, and the distance between points measures their similarity. The goal is then to store P so that one can efficiently find the color (class) c^* most frequently occurring among the k points in P closest to a query point q. See Figure 1(left). We refer to such queries as *chromatic k-NN queries*. To answer such queries, k-NN classifiers often store P in, e.g., a kd-tree and answer queries by explicitly reporting the k points closest to q, scanning through this set to compute the most frequently occurring color [3]. Unfortunately, for many distance measures (including

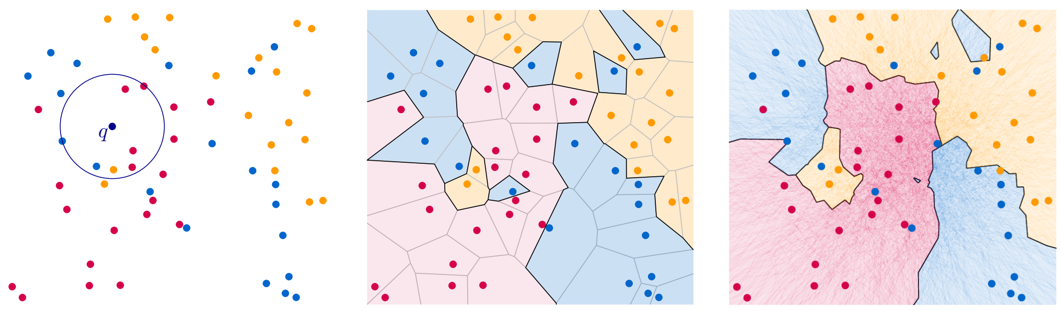

Figure 1 (left) A set of input points from three different classes (colors). The class of a query point q is determined by the labels of its k nearest neighbors (with $k = 7$ as shown here q is classified as red). (middle) The color partition for $k = 1$. (right) The color partition for $k = 3$.

the Euclidean distance) such an approach has no guarantees on the query time other than the trivial $O(n)$ time bound. Even assuming that the dependency on n during the query time is small (e.g. when the points are nicely distributed [11]), the approach requires $\Theta(k)$ time to explicitly process all k points closest to q, whereas the desired output is only a single value: the most frequently appearing color. Hence, our main goal is to design a data structure to store P that has sublinear query time in terms of both n and k, while still using only small space. We focus our attention on the L_2 (Euclidean) distance, and the L_∞ distance metrics. Most of our ideas extend to more general distance measures and higher dimensions as well. However, already in the restricted settings presented here, designing data structures that provide guarantees on the space and query time turns out to be a challenging task.

If the value k is fixed in advance, one possible solution is to build the k^{th}-order Voronoi diagram $Vor_k(P)$ of P, and preprocess it for point location queries. The *k^{th}-order Voronoi diagram* is a partition of $\mathbb{R}^d$ into maximal cells for which all points in a cell (a *Voronoi region*) $V_{k,P}(S)$ have the same set S of k closest points from P. Hence, in each Voronoi region there is a fixed color that occurs most frequently among S. See Figure 1(right) for an illustration. By storing $Vor_k(P)$ in a data structure for efficient (i.e. $O(\log n)$ time) point location queries we can also answer chromatic k-NN queries efficiently. However, unfortunately $Vor_k(P)$ may have size $\Theta(k(n-k))$ [15] (for points in $\mathbb{R}^2$ and the L_2 distance). For other L_m distances the diagram is similarly large [17, 6]. Hence, we are interested in solutions that use less, preferably near-linear, space.

The only result on the theory of chromatic k-NN queries that we are aware of is that of Mount et al. [20]. They study the problem in the case that we measure distance using the Euclidean metric and that the number of colors c, as well as the parameter k, are small constants. Mount et al. state that it is unclear how to obtain a query time independent of k, and instead analyze the query times in terms of the *chromatic density* ρ of a query q. The chromatic density is a term depending on the distance from the query point q to the k^{th} nearest neighbor of q, and the distance from q to the first point at which the answer of the query would change (e.g. the $(k+t)^{\text{th}}$ nearest neighbor of q for some $t > 0$). The intuition is that if many points near q have the same color, queries should be easier to answer than when there are multiple colors with roughly the same number of points. The chromatic density term models this. Their main result is a linear space data structure for points in $\mathbb{R}^d$ that supports $O(\log^2 n + (1/\rho)^d \log(1/\rho))$ time queries. We aim for bounds only in terms of combinatorial properties (i.e. n, c, and k) and allow the number of colors, as well as the parameter k, to be comparable to n. Our results are particularly relevant when k and c are large compared to n.

Table 1 Our results for exact chromatic k-nearest neighbors problems. Bounds marked with * are expected bounds. The general L_m metric bounds hold for $m = O(1)$.

Dimension	Metric	Preprocessing time	Space	Query time
$d = 1$	L_m	$O(n^{3/2} \log n)$	$O(n)$	$O(n^{1/2} \log n)$
$d = 2$	L_m	$\tilde{O}(n^{2/(4+\delta)})$	$O(n)$	$\tilde{O}(n^{1-1/(12+3\delta)})$
$d \geq 3$	L_m	$\tilde{O}(n^{2-1/d} + n^{1+d/(2d-2+\delta)})$	$O(n)$	$\tilde{O}(n^{1-1/((d+1)(2d-2+\delta))})$
$d \geq 2$	L_∞	$O(n^{1+d/(d+1)})$	$O(n)$	$O(n^{1-1/(d+1)+\delta})$
		$\tilde{O}(n^{1+d/(d+1)})$	$O(n \log^{d-1} n)$	$O((n \log^{d-1} n)^{1-1/(d+1)})$
	L_2	$\tilde{O}(n^{2-1/d} + n^{1+d/(d+1)})^*$	$O(n)$	$\tilde{O}(n^{1-(d-1)/d(d+1)})$

Table 2 Our results for the approximate chromatic k-nearest neighbors problems.

Dimension	Metric	Preprocessing time	Space	Query time
$d = 1$	L_m	$O(n \log_{\frac{1}{1-\varepsilon}} n)$	$O(n\varepsilon^{-1})$	$O(\log n + \log \log_{\frac{1}{1-\varepsilon}} n)$
$d = 2$	L_∞	$\tilde{O}(n\varepsilon^{-6})^*$	$\tilde{O}(n\varepsilon^{-6})$	$\tilde{O}(n^{1/2}\varepsilon^{-5/2})^*$
	L_2	$\tilde{O}(n^{1+\delta} + n\varepsilon^{-7})^*$	$\tilde{O}(n\varepsilon^{-4})$	$\tilde{O}(n^{1/2}\varepsilon^{-3/2})^*$

Our approach. Our main idea is to answer a query in two steps. (1) We identify a region $\mathcal{D}^k_m(q)$ that contains exactly the set k-NN$_m(q)$ of the k sites closest to q according to distance metric L_m. (2) We then find the *mode color* c^*; that is, the most frequently occurring color among the points in the region $\mathcal{D}^k_m(q)$. This way, we never have to explicitly enumerate the set k-NN$_m(q)$. We will design separate data structures for these two steps. Our data structure for step (1) will find the smallest metric disk $\mathcal{D}^k_m(q)$ containing k-NN$_m(q)$ centered at q. We refer to such a query as an *range finding* query. If the distance used is clear from the context we may write k-NN(q) and $\mathcal{D}^k(q)$ instead.

Range mode queries. The data structure in step (2) answers so-called *range mode* queries. For these data structures we exploit and extend the result of Chan et al. [8]. They show an array A with n entries can be stored in a linear space data structure that allows reporting the mode of a query range (interval) $A[i..j]$ in $O(n^{1/2})$ time. Furthermore, points in $\mathbb{R}^d$ can be stored in $O(n \operatorname{polylog} n + r^{2d})$ space so that range mode queries with axis-aligned orthogonal ranges can be answered in $O((n/r) \operatorname{polylog} n)$ time. Here, $r \in [1, n]$ is a user-choosable parameter. In particular, setting $r = \lceil n^{1/2d} \rceil$ yields an $O(n \operatorname{polylog} n)$ space solution with $O(n^{1-1/2d} \operatorname{polylog} n)$ query time. Range mode queries with halfspaces can be answered in $O((n/r)^{1-1/d^2} + \operatorname{polylog} n)$ time using $O(nr^{d-1})$ space [8]. Range mode queries in arrays have also been considered in an approximate setting [7]. The goal is then to report an element that appears sufficiently often in the range.

Results and organization. Refer to Table 1 for an overview of our exact solutions. We first consider the problem for n points in $\mathbb{R}^1$. In this setting, we develop an optimal linear space data structure that can find $\mathcal{D}^k_m(q)$ in $O(\log n)$ time, for any $m \geq 1$ (Section 2). We then use Chan et al. [8]'s data structure to report the mode color in $\mathcal{D}^k_m(q)$. Since we present all of our data structures in the pointer machine model augmented with real-valued arithmetic, this yields an $O(n^{1/2}\log n)$ time query algorithm. There is a conditional $\Omega(n^{1/2-\delta})$ lowerbound for chromatic k-NN queries using linear space and $O(n^{3/2})$ preprocessing time (refer to Section 5) so this result is likely near-optimal. In Section 3 we present our data structure for finding $\mathcal{D}^k_m(q)$ in $\mathbb{R}^2$. For the L_∞ metric we show that we can essentially find $\mathcal{D}^k_m(q)$ using a binary search on the radius of the disk, and thus there is a simple $O(n\log n)$ size data structure that allows us to find the range $\mathcal{D}^k(q)$ in $O(\log^2 n)$ time (or $O(n^\delta)$ time, for an arbitrarily small $\delta > 0$, in case of a linear space structure). For the L_2 metric, we can no longer easily access a discrete set of candidate radii. It is tempting to therefore replace the binary search by a parametric search [18, 19]. However, the basic such approach squares the $\tilde{O}(n^{1/2})$ time required to solve the decision problem and is thus not applicable. The full strategy requires a way to generate independent comparisons; typically by designing a parallel decision algorithm [18]. Neither task is straightforward to achieve. Instead, we show that we can directly adapt the query algorithm for answering semi-algebraic range queries [2] (essentially the decision algorithm in the approach sketched above) to find $\mathcal{D}^k_2(q)$ in $O(n^{1/2}\operatorname{polylog} n)$ time. Unfortunately, the final query time for chromatic k-NN queries is dominated by the $O(n^{5/6}\operatorname{polylog} n)$ time range mode queries, for which we use a slight variation of the data structure of Chan et al. [8]. We briefly discuss these results in Section 4, We do show that the data structure can be constructed in $O(n^{5/3})$ expected time, rather than the straightforward $O(n^2)$ time implementation that directly follows from the description of Chan et al.. For the L_∞ metric we can answer range mode queries in $O(n^{3/4}\operatorname{polylog} n)$ time using Chan et al.'s data structure for orthogonal ranges. However, we show that we can exploit that the ranges are squares and use a cutting-based approach (similar to the one used for the L_2 distance) to answer queries in $O(n^{2/3}\log^{2/3} n)$ time instead. If we wish to reduce the space from $O(n\log n)$ to linear the query time becomes $O(n^{2/3+\delta})$.

Since the query times are still rather large, we then turn our attention to approximations; refer to Table 2 for an overview. If f^* is the frequency of the mode color among k-NN(q) then our data structures may return a color that appears at least $(1-\varepsilon)f^*$ times. In case of the L_2 distance our data structure presented in Section 6 now achieves roughly $\tilde{O}(n^{1/2}\varepsilon^{-3/2})$ query time, where the $\tilde{O}$ notation hides polylogarithmic factors of n and ε. The main idea is that approximate levels in the arrangement of distance functions have relatively low complexity, and thus we can store them to efficiently answer approximate range mode queries.

2 The one-dimensional problem

In this section we consider the case where P is a set of n points in $\mathbb{R}^1$. In this case all L_m-distance metrics with $m \geq 1$ are the same, i.e. $L_m(a,b) = |a-b|$. We develop a linear space data structure supporting queries in $O(n^{1/2}\log n)$ time. Building the data structure will take $O(n^{3/2})$ time. We follow the general two-step approach sketched in Section 1. We first show that there is an optimal, linear-space data structure with which we can find $\mathcal{D}^k(q)$ in $O(\log n)$ time, even if k is part of the query. As we then briefly describe how we can directly use the data structure by Chan et al. [8] to find the mode color of the points in $\mathcal{D}^k(q)$ in $O(n^{1/2}\log n)$ time.

Range finding queries. Given a set P of n points in $\mathbb{R}^1$, we wish to store P so that given a query, consisting of a point $q \in \mathbb{R}^1$ and a natural number k, we can efficiently find the k^{th} furthest point from q, and thus $\mathcal{D}^k_m(q)$. We show that by storing P in sorted order in an appropriate binary search tree, we can answer such queries in $O(\log n)$ time. To this end, we first consider answering so called rank queries on a ordered set, represented by a pair of binary search trees. This turns out to be the crucial ingredient in the data structure.

Let $R \cup B$ be an ordered set of elements, and let T_R be a be a binary search tree whose internal nodes store the elements from R, and in which each node is annotated with the number of elements in that subtree. Similarly, let T_B be a binary search tree storing B. We can efficiently compute the element among $R \cup B$ with rank k (i.e. the k^{th} smallest element):

▶ **Lemma 2.1.** *Let T_B and T_R be two binary search trees with size annotations, and let k be a natural number. We can compute the element of rank k among $B \cup R$ in $O(height(T_B) + height(T_R))$ time.*

To support efficiently querying $\mathcal{D}^k(q)$ we now store P in a balanced binary search tree T_P with subtree-size annotations, so that we can: (i) search for the element of rank k, and (ii) given a query value $q \in \mathbb{R}^1$ we can split the tree at q in $O(\log n)$ time. We can implement T_P using e.g. a red black tree [22] (although with some care even a simple static balanced binary search tree will suffice). We will use the split operations only to answer queries; so we use path copying in this operation, so that we can still access the original tree once a query finishes [21]. The data structure uses $O(n)$ space, and can be built in $O(n \log n)$ time.

Given a query (q, k) the main idea is now to split T_P into two trees $T_{P^<}$ and $T_{P^\geq}$, where $P^< \subseteq P$ is the set of points left of q and $P^\geq = P \setminus P^<$ is the remaining set of points right of q (or coinciding with q). Observe that in these two trees, the points are actually ordered by distance to q (albeit for $T_{P^<}$ the points are stored in decreasing order while the points in $T_{P^\geq}$ are stored in increasing order). So, we can essentially use the procedure from Lemma 2.1 on the trees $T_{P^<}$ and $T_{P^\geq}$ to find the point in $P^\leq \cup P^>$ with rank k (according to the "by distance to q"-order). The only difference with the algorithm as described in Lemma 2.1 is that for $T_{P^<}$ the roles of the left and right subtree are reversed. Splitting T into $T_{P^<}$ and $T_{P^\geq}$ takes $O(\log n)$ time. Since both subtrees have height at most $O(\log n)$, Lemma 2.1 also takes $O(\log n)$ time. So, we obtain the following result:

▶ **Theorem 2.2.** *Let P be a set of n points in $\mathbb{R}^1$. In $O(n \log n)$ time we can build a linear space data structure so that given a query q, k we can find the smallest disk $\mathcal{D}^k_m(q)$, with respect to any L_m metric, containing k-NN$_m(q)$ in $O(\log n)$ time.*

Range mode queries. What remains is to store P such that given a query interval Q we can efficiently report the mode color among $P \cap Q$. Chan et al. [8] show that an array A of size n can be preprocessed in $O(n^{3/2})$ time into a linear space structure that can report the range mode of a query range $A[i : j]$ in $O(n^{1/2})$ time. By implementing arrays with balanced binary search trees, we can also implement this structure in the pointer machine model. This increases the query and preprocessing times by an $O(\log n)$ factor. We then store (the colors of) the points in increasing order in this structure. Together with Theorem 2.2 we obtain:

▶ **Theorem 2.3.** *Let P be a set of n points in $\mathbb{R}^1$. In $O(n^{3/2} \log n)$ time, we can build a data structure of size $O(n)$, that answers chromatic k-NN queries on P in $O(n^{1/2} \log n)$ time.*

3 Range finding queries two dimensions

In this section we give a data structure that, given a query point $q \in \mathbb{R}^2$, reports the smallest range $\mathcal{D}^k_m(q)$ centered at q containing k-NN$_m(q)$. In Section 3.1 we consider the case where the L_∞ metric is used. In Section 3.2 we then consider the L_2 metric.

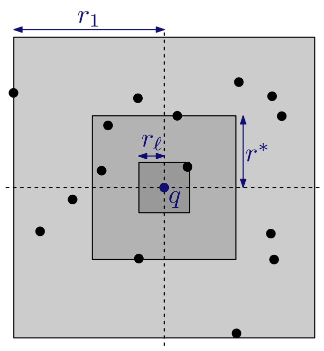

Figure 2 The considered radii r_i, and r^*, for $k = 5$.

3.1 The L_∞ metric case

For a given value $r \geq 0$, define $S(q, r) = [q_x - r, q_x + r] \times [q_y - r, q_y + r]$ as the axis-aligned square with sidelength $2r$ centered at q. We call r the *radius* of such a square. Now observe that $\mathcal{D}^k(q) = S(q, r^*)$ is also an axis-aligned square, in particular with radius r^* equal to the distance $L_\infty(q, p)$ between q and the k^{th} nearest neighbor p of q. Thus we either have $r^* = |q_x - p_x|$ or $r^* = |q_y - p_y|$. This leads us to the following data structure. We store the x-coordinates $x_1, \ldots, x_n$ of the points in P in increasing order in a balanced binary search tree. Similarly, we store the y-coordinates of the points in P in sorted order $y_1, \ldots, y_n$. We set $x_0 = y_0 = -\infty$ and $x_{n+1} = y_{n+1} = \infty$, and call these values coordinates as well. In addition, we store P in a data structure for $O(\log n)$ time orthogonal range counting queries, for which we use a range tree [5, 23]. The entire data structure can be constructed in $O(n \log n)$ time, and uses $O(n \log n)$ space.

Now let $x_0, \ldots, x_\ell$ be the x-coordinates that are at most q_x. The sequence $r_i = |q_x - x_i|$, for $i = 0, \ldots, \ell$, defines a sequence of decreasing radii. See Figure 2 for an illustration. We can find the smallest radius r_i for which $S(q, r_i)$ contains at least k points by performing binary search over the radii, performing orthogonal range counting at each step to guide the search. By performing a similar procedure for the x-coordinates greater than q_x, as well as for the y-coordinates, we obtain a set of four squares, that each contain at least k points. The smallest of these squares contains exactly k points and is thus $\mathcal{D}^k(q)$. As each procedure performs $O(\log n)$ orthogonal range counting queries, we obtain the following theorem.

▶ **Theorem 3.1.** *Let P be a set of n points in $\mathbb{R}^2$. In $O(n \log n)$ time, we can build a data structure of size $O(n \log n)$, that can report $\mathcal{D}_1^k(q)$ and $\mathcal{D}_\infty^k(q)$ in $O(\log^2 n)$ time.*

Following an idea of Chan et al. [8] we can reduce the space used by replacing the binary range tree used for the orthogonal range queries by one with fanout n^δ for some constant $\delta > 0$. This increases the query time to $O(n^\delta + k')$ time, where k' is the output size.

▶ **Theorem 3.2.** *Let P be a set of n points in $\mathbb{R}^2$. Let $\delta > 0$ be an arbitrarily small constant. In $O(n \log n)$ time, we can build a data structure of $O(n)$ size, that can report $\mathcal{D}_1^k(q)$ and $\mathcal{D}_\infty^k(q)$ in $O(n^\delta)$ time.*

3.2 The L_2 metric case

3.2.1 Randomized query times

In this section, we sketch a simple randomized data structure that finds $\mathcal{D}_2^k(q)$ in expected $O(n^{1/2}\log^{B+1} n)$ time[1]. The data structure consists of the large fan-out partition tree of Agarwal et al. [2], used for semialgebraic range searching. Together with this tree, we take a random sample P' of P by including each point with probability $1/n^{1/2}$. Note that this random sample will contain $n^{1/2}$ points in expectation.

The main idea is to combine binary search on the ordered distances $R = \{L_2(q, p') \mid p' \in P'\}$ with circular range counting. Let r^* be the distance between q and its k^{th} nearest neighbor among P. We then search for two consecutive distances $r_i, r_{i+1} \in R$, such that $r_i \leq r^* \leq r_{i+1}$. Because the number of points $p \in P$ with $r_i \leq L_2(q, p) \leq r_{i+1}$ is $n^{1/2}$ in expectation, we can then afford to report these points with semialgebraic range reporting, and combining binary search with range counting again to find r^*. This leads to an expected query time of $O(n^{1/2}\log^{B+1} n)$.

3.2.2 Worst-case query times

In this section we show how to achieve the same complexity bounds of Section 3.2.1, but with a worst-case query time bound, rather than a randomized bound.

The data structure. For now, we assume that the set P lies in D_0-general position for some constant D_0 (see [2] for a definition). The details on this assumption are not important, and we show in the full version of the paper how to handle arbitrary point sets. The data structure consists of two copies of the large fan-out (fan-out n^δ, for some constant $\delta > 0$) partition tree of Agarwal et al. [2], built on P. The first copy, which we call $\mathcal{T}$, will be augmented slightly to support generating candidate ranges (disks) that will eventually lead to $\mathcal{D}^k(q)$. The second copy will be used as a black box, answering circular range counting queries to guide the search for $\mathcal{D}^k(q)$ by counting the number of points inside the candidate ranges.

The tree $\mathcal{T}$ is constructed by recursively partitioning the space into open, connected regions, called *cells*. Once a cell contains a small (constant) number of points of P, the recursion stops and $\mathcal{T}$ gets a leaf node containing these points. There may be points of P that do not lie on these cells, but rather on the zero set of the partitioning polynomial used to partition the space. For range searching with arbitrary point sets, Agarwal et al. [2] store these points in an auxiliary data structure. However, with our assumption that P lies in D_0-general position, we do not need this auxiliary data structure, and will simply store the points inside a leaf node, whose parent is the node corresponding to the partitioning polynomial. We further adjust $\mathcal{T}$ such that each internal node corresponding to a cell ω stores an arbitrary point in ω. During the construction of $\mathcal{T}$, a point inside each cell is already computed. Hence the construction time is unaltered.

Answering a query. To query the structure with a query point q, we keep track of a set of nodes N_i for each level i of $\mathcal{T}$ that is explored by our algorithm. With slight abuse of terminology, we refer to the sets P' of points stored in the leaves of $\mathcal{T}$ as cells. Let Ω_i denote

[1] We would like to thank an anonymous reviewer for the solution sketched in this section, which led to our current solution for finding $\mathcal{D}_2^k(q)$ in Section 3.2.

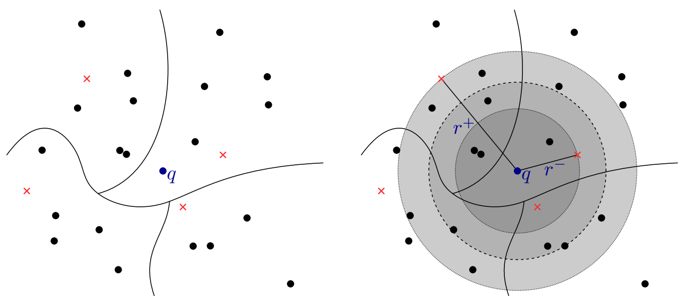

Figure 3 (left) A partitioning of P into four cells. The red crosses are the points $p(\omega)$. (right) The disk $\mathcal{D}^k(q)$ (dashed boundary) and the disks $D(q, r^-)$ (dotted boundary, dark gray) and $D(q, r^+)$ (dotted boundary, light gray).

the cells corresponding to the internal nodes in N_i, and P_i denote the cells corresponding to the leaf nodes in N_i. We maintain that $p^k(q)$, the k^{th} nearest neighbor of q, is contained in a cell in $\Omega_i \cup P_i$. Initially, N_1 contains the root node. If $\mathcal{T}$ is a single leaf, then the cell corresponding to the root node will be stored in P_1. Otherwise, it will be stored in Ω_1.

The query algorithm works as follows. Say the algorithm is at level i in $\mathcal{T}$. For a cell $\omega \in \Omega_i$ stored in an internal node, let $p(\omega)$ be the point inside ω that was stored with it. Let

$$R_i = \{L_2(q, p(\omega)) \mid \omega \in \Omega_i\} \cup \bigcup_{P_i' \in P_i} \{L_2(q, p) \mid p \in P_i'\}$$

be the set of distances to these points $p(\omega)$, as well as to all points stored in the leaves in P_i and in the nodes in N_i. Let r^* be the distance between q and its k^{th} nearest neighbor among P. This is the radius of $\mathcal{D}^k(q)$. To find this radius, we compute the largest distance $r^- \in R_i$, and the smallest distance $r^+ \in R_i$, such that $r^- < r^* \leq r^+$. See Figure 3. If r^- (respectively r^+) does not exist, set it to 0 (respectively ∞). We show how to compute r^+. Computing r^- works similarly.

To compute r^+, we use a combination of median finding and binary search. For some radius $r \in R_i$, we decide if $r^+ > r$ or $r^+ \leq r$ using a circular counting query. If $D(q, r)$ contains less than k points, we have $r^+ > r$. Otherwise, we have $r^+ \leq r$. By performing this procedure for the median radius in R_i, we can discard half of R_i with each query.

Once we have that r^+ is the distance between q and a point in P (it is constructed through a leaf node of $\mathcal{T}$), we have found r^*. We then terminate the algorithm, returning the disk with the found radius. Otherwise we continue the search in the next level of $\mathcal{T}$. To continue the search through the tree, we construct the set N_{i+1} by replacing every node $\nu \in N_i$ with its children whose cells are crossed by one of $D(q, r^-)$ and $D(q, r^+)$. A cell ω is *crossed* by a disk D if $\omega \cap D \neq \emptyset$ and $\omega \nsubseteq D$. The sets Ω_{i+1} and P_{i+1} are then constructed from these child nodes. Once these sets are constructed, we advance the algorithm to level $i + 1$ and repeat the procedure.

▶ **Lemma 3.3.** *The query algorithm correctly returns $\mathcal{D}^k(q)$.*

Proof. We claim that this algorithm correctly returns $\mathcal{D}^k(q)$. First, note that if the algorithm returns a disk, that disk contains k points, and its radius is equal to the distance between q and a point of P. Therefore, it is indeed $\mathcal{D}^k(q)$. We now show that our algorithm will always return a disk, and therefore that our algorithm is correct. To show this, it suffices to show that for every level i of $\mathcal{T}$ traversed by our algorithm, the set $\Omega_i \cup P_i$ contains the cell containing $p^k(q)$.

We give a proof by induction. It holds trivially that $\Omega_1 \cup P_1$ contains a cell containing $p^k(q)$. We prove that if $\Omega_i \cup P_i$ contains a cell ω' containing $p^k(q)$, then either the algorithm terminates and returns $\mathcal{D}^k(q)$, or there is a cell $\omega \in \Omega_{i+1} \cup P_{i+1}$ containing $p^k(q)$.

If $\omega' \in P_i$, then R_i will contain r^*. It will then find $r^+ = r^*$ and terminate, returning $D(q, r^+) = \mathcal{D}^k(q)$. Now assume that $\omega' \in \Omega_i$. Let $\nu \in N_i$ be the internal node corresponding to ω'. Now let ω be the cell stored in a child of ν, such that $p^k(q)$ lies in ω. We show that $\omega \in \Omega_{i+1} \cup P_{i+1}$.

Our algorithm performs repeated median finding on the radii in R_i, resulting in the largest radius $r^- \in R_i$ and smallest radius $r^+ \in R_i$, such that $D(q, r^-)$ and $D(q, r^+)$, contains less than, respectively at least, k points of P. Let r_ω be the distance between q and $p(\omega)$, the point in ω that was stored in $\mathcal{T}$. If $r_\omega < r^*$, then we have that $r_\omega \le r^-$, implying that $D(q, r^-)$ intersects ω. Also, because ω is open, and because $p^k(q) \in \omega$, there must be a point $p' \in \omega$ such that $r^- < r^* = L_2(q, p^k(q)) < L_2(q, p')$. This shows that ω is not contained in $D(q, r^-)$, and thus that $D(q, r^-)$ crosses ω. With similar reasoning, it can be seen that if $r_\omega \ge r^*$, then $D(q, r^+)$ crosses ω. Thus, ω will be crossed by at least one of $D(q, r^-)$ and $D(q, r^+)$, and thus $\omega \in \Omega_{i+1} \cup P_{i+1}$. Hence, our algorithm is correct. ◀

▶ **Theorem 3.4.** *Let P be a set of n points in $\mathbb{R}^2$. Let $\delta > 0$ be an arbitrarily small constant. In $O(n^{1+\delta})$ expected time, we can build a data structure of $O(n)$ size, that can report $\mathcal{D}^k_2(q)$ in $O(n^{1/2} \operatorname{polylog} n)$ time.*

4 Range mode queries in two dimensions

In this section we discuss answering range mode queries in $\mathbb{R}^2$, and see how we can use them together with the data structures from Section 3 to answer chromatic k-NN queries. Our results build on the data structure of Chan et al. [8] for finding the mode color among three-dimensional points in halfspaces. We show that using a standard lifting transformation that maps the points $P \subseteq \mathbb{R}^2$ into planes in $\mathbb{R}^3$, we can apply their result to answer range mode queries with disks in the L_2 metric. Our main contribution is that we show that a similar approach can answer range mode queries with squares (disks in the L_∞ metric). Somewhat surprisingly, this leads to better query times compared to directly using the existing range mode data structures for orthogonal ranges [8]. Finally, we show that we can, in fact, construct the data structures in (expected) $O(n^{5/3})$ time rather than $O(n^2)$ (worst-case) time. We briefly sketch these ideas here. Refer to the full version for details.

The data structure for the L_2 metric. We lift the set of points $P \subseteq \mathbb{R}^2$ to a set of planes H in $\mathbb{R}^3$. The points in a query disk Q map to the subset of planes passing below a point h^*. Hence, we have to report the mode color c of these planes. The main idea in the data structure of Chan et al. [8] is to build a $(1/r)$-cutting on the planes in H; a subdivision of $\mathbb{R}^3$ into $O(r^3)$ interiorly disjoint cells, whose interiors are each crossed by at most n/r planes (for $r = n^{1/3}$). Let Δ be the cell containing the query point h^*. The key insight is that the mode color c of the planes passing below the query point h^* is either the mode color c_Δ of all planes passing below Δ, or the color of one of the planes that intersect Δ. The data structure stores the color c_Δ for each cell Δ, so at query time we only have to consider the at most n/r colors of the planes that intersect Δ. Our insight is that for k-NN queries we can use the tools from Section 3 to find these colors in $\tilde{O}(n^{1/2})$ time, leading to a total query time of $\tilde{O}(n^{5/6})$ (rather than $\tilde{O}(n^{8/9})$ time in the case of arbitrary query points in $\mathbb{R}^3$).

The data structure for the L_∞ metric. For a point $p \in P \subseteq \mathbb{R}^2$, the graph of the distance function $L_\infty(p,q) = \max\{|p_x - q_x|, |p_y - q_y|\}$ forms an upside-down pyramid ∇_p in $\mathbb{R}^3$. For a point $q \in \mathbb{R}^2$ we have that $L_\infty(p,q) \leq r$ if and only if (q_x, q_y, r) lies above ∇_p. Thus, we can again transform the problem to that of finding the mode color among those graphs in $\mathbb{R}^3$ that lie below a given query point. Using a similar cutting-based solution as sketched above – again exploiting that we can quickly find the (colors of the) graphs intersecting a cell Δ in $\mathbb{R}^2$ rather than $\mathbb{R}^3$ – yields a linear space data structure answering queries in an $O(n^{2/3+\delta})$ time. Using $O(n \log n)$ space we can decrease the query time to $O(n^{5/3} \log^{2/3} n)$.

Construction. Our final contribution in terms of the range mode data structures is that we show how to efficiently construct the above data structures. In case of both the L_2 and the L_∞ metrics, there is a somewhat straightforward algorithm to construct the above data structures in $O(n^2)$ time. The bottleneck being the time required to compute the mode color c_Δ for all cells Δ of the cutting. We argue that this can actually be done in roughly $\tilde{O}(nr^2 + r^3 n^\alpha)$ time, for some $\alpha \in (0,1)$. For our choice of parameter r, this leads to algorithms with subquadratic running time. We summarize our results for the chromatic k-NN problem in the following theorems.

▶ **Theorem 4.1.** *Let P be a set of n points in $\mathbb{R}^2$. There is a linear space data structure that answers chromatic k-NN queries on P with respect to the L_2 metric. Building the data structure takes expected $O(n^{5/3})$ time, and queries take $O(n^{5/6} \operatorname{polylog} n)$ time.*

▶ **Theorem 4.2.** *Let P be a set of n points in $\mathbb{R}^2$. There is a linear space data structure that answers chromatic k-NN queries on P with respect to the L_∞ metric. Building the data structure takes $O(n^{5/3})$ time, and queries take $O(n^{2/3+\delta})$ time. We can decrease the query time to $O(n^{2/3} \log^{2/3} n)$ time using $O(n \log n)$ space and $O(n^{5/3} \log^{2/3} n)$ preprocessing time.*

5 Lower bounds

We now discuss to what extent our results for the exact version of the problem may be improved further. Chan et al. [8] show that there is a conditional $\Omega(n^{1/2-\delta})$ time lower bound on range mode queries, provided we insist on using only linear space and $O(n^{3/2})$ preprocessing time. We extend this bound to chromatic k-NN queries in $\mathbb{R}^1$. More generally, Lemma 5.1 shows that we can reduce chromatic k-NN queries in $\mathbb{R}^d$ to range mode queries using range counting. Due to space restrictions, further details on the lower bound can be found in the full version of the paper.

▶ **Lemma 5.1.** *Let P be a set of n points in $\mathbb{R}^d$. A range mode query with a query ball $\mathcal{D}_m$ (with respect to the L_m metric) can be answered using a single range counting query with query range $\mathcal{D}_m$ and a single chromatic k-NN query.*

Proof. We use the range counting query to find the number of points in the range $\mathcal{D}_m$. Let this be k, and let q be the center point of the ball $\mathcal{D}_m$. Hence, $\mathcal{D}^k_m(q) = \mathcal{D}_m$, and thus the answer to the chromatic k-NN query with center q and value k is the mode color of $\mathcal{D}_m$. ◀

Relations to range counting queries. Next, we relate the cost of range finding queries, i.e. the problem solved in our first step, to range counting queries. Given a data structure for range finding queries we can answer range counting queries using only logarithmic overhead:

▶ **Lemma 5.2.** *Let P be a set of n points in $\mathbb{R}^d$. A range counting query on P with a disk $\mathcal{D}_m$ under metric m can be performed using $O(\log n)$ range finding queries.*

Proof. Let q be the center of the query disk. We binary search over the integers $0, \dots, n$, using a range finding query to find a disk $\mathcal{D}^k_m(q)$ for each considered integer k. If the reported disk is smaller than $\mathcal{D}_m$, the number of points inside $\mathcal{D}_m$ is at least k. Otherwise, the number of points is smaller than k. It follows that with $O(\log n)$ range finding queries, we can count the number of points in $\mathcal{D}_m$. ◀

It thus follows that range finding is roughly as difficult as range counting. In particular, a $Q(n)$ time lower bound for range counting queries using $S(n)$ space implies an $\tilde{\Omega}(Q(n))$ time lower bound for range finding queries with $S(n)$ space. For example, in the semigroup model there is an $\tilde{\Omega}(n/S(n)^{1/d})$ time lower bound for halfspace range counting [4]. Since every halfspace is a disk $\mathcal{D}_2$ (of radius ∞), this lower bound also holds for range counting with disks in the L_2 metric, and thus also for range finding.

The range mode queries from step 2 are also related to a form of range counting. A "type-2" range counting query with query range Q asks for all the distinct colors appearing in Q together with their frequencies, i.e. for each reported color c we must also report the number of points in $P \cap Q$ that have color c [9]. Clearly, answering "type-2" queries is more difficult than range counting (just assign all points the same color), so the above lower bounds also hold for "type-2" queries. Such "type-2" queries however also allow us to solve the range mode problem. When the number of colors is small (e.g. two), and we already know the number of points k in the query range $\mathcal{D}^k(q)$ it seems that answering range mode queries is not much easier than answering ("type-2") range counting queries. We therefore conjecture that answering range mode queries is roughly as difficult as answering range counting queries.

▶ **Conjecture 5.3.** *If answering a range counting query with a query range $\mathcal{D}$ using $S(n)$ space requires $Q(n)$ time then answering a range mode query with query range $\mathcal{D}$ using $S(n)$ space requires $\tilde{\Omega}(Q(n))$ time.*

Note that this conjecture together with Lemma 5.1 would imply that answering a k-NN query is at least as hard as answering a range counting query. Furthermore, since we can answer a range counting query using $O(\log n)$ range finding queries (Lemma 5.2) that would then mean our two-step approach has negligible overhead with respect to an optimal solution to chromatic k-NN queries.

6 The approximate problems

In this section we sketch our approach for answering ε-approximate chromatic k-NN queries. Refer to the full version for details. Our goal is to report a color c that occurs at least $(1-\varepsilon)f^*$ times, where f^* is the frequency of the mode color c^* of k-NN(q). We again use the two-step approach of finding the range $\mathcal{D}^k(q)$ (step (1)) and computing the mode of the range (step (2)). We use exact range finding data structures from Sections 2 and 3, and focus our attention on approximating step (2) for two reasons: first, the running times in our exact solutions are dominated by step (2), and second, it is unclear how to use approximate solutions to k-NN queries (that is, approximate ranges) and still obtain guarantees on the approximation factor of our ε-approximate chromatic k-NN queries.

For points in $\mathbb{R}^1$ we can directly use the result of Bose et al. [7] to answer $(1-\varepsilon)$-approximate range mode queries. In $O(n \log_{\frac{1}{1-\varepsilon}} n)$ time, we can thus build an $O(n/\varepsilon)$ size data structure that can answer k-NN queries in $O(\log n + \log\log_{\frac{1}{1-\varepsilon}} n)$ time.

For points in $\mathbb{R}^2$, and the L_2-metric, we answer approximate range mode queries as follows; we use similar ideas for the L_∞ metric. We use the standard lifting transformation to transform the set of points P a set of planes H. A query disk Q now corresponds to a

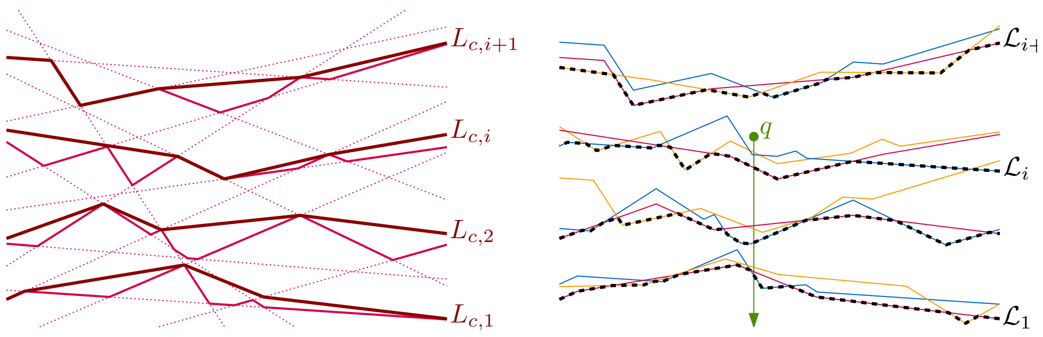

Figure 4 (left) An illustration of the idea in $\mathbb{R}^2$, the planes (here lines) of a single color, their $k_i = \left(\frac{1}{1-\alpha}\right)^i$-levels (bright red), and the $g(\varepsilon)$-approximate k_i-levels $L_{c,0}, L_{c,1}, \ldots$ (in dark red). (right) For each i, the $\mathcal{L}_i$ forms the lower envelope of the $L_{c,i}$ surfaces over all colors c. We search for the largest i for which q lies above $\mathcal{L}_i$ (dashed).

vertical halfline with a top endpoint h^*, and the mode color c^* of $P \cap Q$ is the most frequently occurring color among the planes passing below h^*. Our aim is to report a color c such that at least $(1-\varepsilon)f^*$ planes of that color pass below h^*.

The main idea to answer approximate range mode queries efficiently is to compute, for each color c, a series of $g(\varepsilon)$-approximate k_i-levels (for some function g) considering only the planes of color c. For each choice of i, we then consider the lower envelope $\mathcal{L}_i$ of all those k_i-levels among the various colors. See Figure 4 for an illustration. Now observe that if h^* lies in between $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$, the frequency f^* of a mode color c^* may only be a $g(\varepsilon)$ fraction larger than k_{i+1}, while the frequency of the color defining $\mathcal{L}_i$ directly below h^* is at least k_i. So if $g(\varepsilon)$ and k_i/k_{i+1} are sufficiently small this is a $(1-\varepsilon)$-approximation. One additional complication is that even though our $g(\varepsilon)$-approximate k_i-levels have fairly small complexity, their lower envelopes do not. So, we need to design a data structure that can test if h^* lies above or below $\mathcal{L}_i$ without explicitly storing $\mathcal{L}_i$. We show that with near-linear space we can answer such queries in $O_\varepsilon(n^{1/2})$ time. This then leads to an $\tilde{O}_\varepsilon(n^{1/2})$ time query algorithm for answering k-NN queries.

We use the same approach to answer approximate queries under the L_∞ metric. Here, the approximate levels are constructed using the result of Kaplan et al. [13]. This leads to roughly the same complexities.

7 Concluding Remarks

We presented the first data structures for the chromatic k-NN problem with query times that depend only on the number of stored points. While we focused mostly on the two-dimensional case, our exact result extend to higher dimensions and other metrics as well (see full version). Our two-step approach essentially reduces the problem to efficiently answering range mode queries. The main open question is how to answer such queries efficiently. Since it is unlikely that we can answer such queries in $\Omega(n^{1/2})$ time (using only near-linear space), it is also particularly interesting to consider further improvements to the ε-approximate query data structures. For the Euclidean distance, finding the query range may now be the dominant factor (depending on the choice of ε). One option is to report a range that contains only approximately the k nearest neighbors of a query point. However, this further complicates the analysis. For the L_∞ distance it may also be possible to reduce the space usage by using a different method for computing the approximate levels (e.g. using the results by Agarwal et al. [1] or the recent results of Liu [16]).

References

1 Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29:912–953, 1999.

2 Pankaj K. Agarwal, Jirí Matousek, and Micha Sharir. On range searching with semialgebraic sets. II. *SIAM J. Comput.*, 42(6):2039–2062, 2013. `doi:10.1137/120890855`.

3 Charu C Aggarwal. *Data classification: algorithms and applications*. CRC press, 2014.

4 Sunil Arya, David M. Mount, and Jian Xia. Tight lower bounds for halfspace range searching. *Discrete & Computational Geometry*, 47(4):711–730, 2012. `doi:10.1007/s00454-012-9412-x`.

5 Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980. `doi:10.1145/358841.358850`.

6 Cecilia Bohler, Panagiotis Cheilaris, Rolf Klein, Chih-Hung Liu, Evanthia Papadopoulou, and Maksym Zavershynskyi. On the complexity of higher order abstract voronoi diagrams. *Computational Geometry*, 48(8):539–551, 2015. `doi:10.1016/j.comgeo.2015.04.008`.

7 Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate range mode and range median queries. In Volker Diekert and Bruno Durand, editors, *STACS*, pages 377–388, 2005. `doi:10.1007/978-3-540-31856-9_31`.

8 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.

9 Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further Results on Colored Range Searching. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SoCG.2020.28`.

10 Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

11 Jerome H. Friedman, Jon Louis Bentley, and Raphael A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977. `doi:10.1145/355744.355745`.

12 W. E. Henley and D. J. Hand. A k-nearest-neighbour classifier for assessing consumer credit risk. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 45(1):77–95, 1996. URL: `http://www.jstor.org/stable/2348414`.

13 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64:838–904, 2020.

14 Yan-Nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 108–120, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

15 D. T. Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Transactions on Computing*, 31:478–487, 1982.

16 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2842–2859. SIAM, 2020. `doi:10.1137/1.9781611975994.173`.

17 Chih-Hung Liu, Evanthia Papadopoulou, and Der-Tsai Lee. The k-nearest-neighbor voronoi diagram revisited. *Algorithmica*, 71(2):429–449, 2015. `doi:10.1007/s00453-013-9809-9`.

18 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

19 Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979. `doi:10.1287/moor.4.4.414`.

20 D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. Chromatic nearest neighbor searching: A query sensitive approach. *Computational Geometry*, 17:97–119, 2000.

21 Neil Sarnak and Robert E Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

22 Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

23 Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985. doi:10.1137/0214019.