Chromatic distribution of k-nearest neighbors of a line segment in a planar colored point set

Partha P. Goswami [a], Sandip Das [b], Subhas C. Nandy [b,*]

[a] *Calcutta University, Calcutta 700 0073, India*
[b] *Indian Statistical Institute, Calcutta 700 108, India*

Received 19 September 2005

Available online 9 January 2007

Communicated by F. Dehne

Abstract

Let P be a set of n colored points distributed arbitrarily in $\mathbb{R}^2$. The chromatic distribution of the k-nearest neighbors of a query line segment ℓ is to report the number of points of each color among the k-nearest points of the query line segment. While solving this problem, we have encountered another interesting problem, namely the *semicircular range counting query*. Here a set of n points is given. The objective is to report the number of points inside a given semicircular range. We propose a simple algorithm for this problem with preprocessing time and space complexity $O(n^3)$, and the query time complexity $O(\log n)$. Finally, we propose the algorithm for reporting the *chromatic distribution* of k nearest neighbors of a query line segment. Using our proposed technique for *semicircular range counting query*, it runs in $O(\log^2 n)$ time.

1. Introduction

Let P be a set of n points arbitrarily distributed in $\mathbb{R}^2$. Each point is colored with any one of m given colors ($1 \leqslant m \leqslant n$). In the chromatic version of the nearest neighbor problem, for a given query point q, the objective is to answer the following queries among its k nearest neighbors of q: (Q1) the color that occurs most frequently, and (Q2) the number of distinct colors. Different variations of *chromatic nearest neighbors problem* are studied by many researchers [4,5, 7–10]. Of these, in [4,8,9], the problem of answering the query Q1 is studied in the context of practical applications, and not from the perspective of worst case asymptotic complexity analysis. The reason is that it is not clear how to determine the most common color among the k nearest neighbors without explicitly observing all k nearest neighbors of that point [10]. In the same paper, an algorithm is also proposed for answering the query Q1 under the assumption that the color classes form spatially well separated clusters. Assuming that the number of color classes is constant, the proposed algorithm can preprocess the point set P in $O(n \log n)$ time using $O(n)$ space, and can answer the query in $O(\log^2 n + (\frac{1}{\delta})^2 \log(\frac{1}{\delta}))$ time, where δ is a measure of the well-separateness of the color classes. The value of δ increases with spatial separation of the color classes. Gupta et al. [7] considered the query problem Q2. Graf and Hinrichs [5] studied another version of the chromatic nearest neighbor problem, where the query point

is associated with a color and the problem is to find the nearest neighbor of a different color.

We consider a general version of the *chromatic k nearest neighbors problem* where the query object is an arbitrary straightline segment. Here, the number of colors m is assumed to be a constant, but k may be an input along with the query line segment. Except the general position assumption, no other assumptions have been made about the given point set. Our objective is to count the number of points of each color i, $1 \leqslant i \leqslant m$. The preprocessing time and space complexities of our method are both $O(n^3)$, it can answer an arbitrary query in $O(\log^2 n)$ time.

Along the way to solving this problem, we have encountered an interesting range searching problem, namely the *semicircular range counting query* problem in $\mathbb{R}^2$. It asks to preprocess the given set P of points in a suitable data structure which can efficiently count the number of points in P that lie inside any arbitrary query semicircle without inspecting all of them.

We propose an algorithm for the semicircular range counting problem. Its time and space complexities for preprocessing are both $O(n^3)$, and the query time complexity is $O(\log n)$. This matches with the complexity results of the classical circular range counting query problem [1,2]. The semicircular range counting query may find its applications in many other problems.

2. Problem formulation

Let $\sigma = [a, b]$ be a line segment with a and b as its end points. Distance of a point p from σ is defined as follows:

From p, draw perpendicular on the line containing the line segment σ. If it intersects σ properly (at a point, say q), then the distance from p to σ is $d(p, q)$, where $d(.,.)$ denotes the Euclidean distance between two points. Otherwise, it is $\min(d(p, a), d(p, b))$.

The shape of the region R containing a specified number (k) of nearest neighbors of σ is like a hippodrome as shown in Fig. 1, and its width w is determined by the farthest among its k nearest neighbors. Obviously, width of this region is monotonically increasing function of the value of k. The region R can be split into three parts as follows:

R_{mid}: the rectangular region of width $2w$, where the line-segment σ joins the mid-point of one pair of its parallel edges ℓ_1 and ℓ_2, each of length w, and

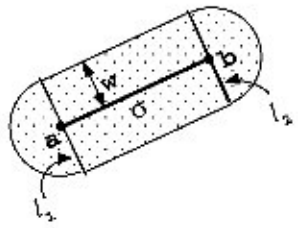

Fig. 1. Region containing δ-nearest neighbors of a line segment.

R_a and R_b: two semicircles, each of radius w, with centers at a and b, respectively, and diameters aligned with ℓ_1 and ℓ_2, respectively.

The subset of points which are properly inside R_{mid} (not on ℓ_1 and ℓ_2) is denoted as Δ_{mid}, and those in R_a and R_b (including those in ℓ_1 and ℓ_2, respectively), are denoted as Δ_a and Δ_b, respectively. Note that, $|\Delta_{\text{mid}}| + |\Delta_a| + |\Delta_b| = k$.

Our solutions for the two variations of the chromatic distribution of k nearest neighbors problem for a line-segment are based on the algorithms of the following two subproblems:

Rectangular range counting query: Given a line-segment σ and a real number w, report the number of points inside the rectangle of width w whose one side is aligned to σ.

Semicircular range counting query: Given a line segment ℓ, report the number of points lying within the specified semicircle with ℓ as diameter.

The *rectangular range counting query* can be answered in $O(\log n)$ time with $O(n^2)$ preprocessing time and space [6]. Here, we develop an algorithm for the *semicircular range counting query* which performs in $O(\log n)$ time, but the preprocessing time and space are both $O(n^3)$.

3. Geometric preliminaries

We use geometric duality for solving the problems mentioned above. A point $p = (a, b, c) \in \mathbb{R}^3$ is mapped to a dual plane $\mathcal{D}(p)$: $z = ax + by - c$, and conversely the image $\mathcal{D}(z)$ of such a plane $z = ax + by - c$ is the point $p = (a, b, c)$. Notice that a point p is above (resp., on, below) a non-vertical plane z if and only if $\mathcal{D}(p)$ is below (resp., on, above) $\mathcal{D}(z)$. For the set of points $P \in \mathbb{R}^3$, the corresponding dual planes are denoted by $\mathcal{D}(P)$. The arrangement of the planes in $\mathcal{D}(P)$, denoted by $\mathcal{A}(\mathcal{D}(P))$, decomposes the entire space into convex

cells of dimensions 0, 1, 2, 3. If the points in $P \in \mathbb{R}^3$ are in general position, then the total number of cells in $\mathcal{A}(\mathcal{D}(P))$ is $O(n^3)$ [3], where $n = |P|$.

Result 1. Given a set of non-vertical parallel planes in the primal space (in $\mathbb{R}^3$), the corresponding points in the dual space lie along a line perpendicular on the xy-plane.

In the arrangement $\mathcal{A}(\mathcal{D}(P))$ in $\mathbb{R}^3$, a point q is said to be at level θ $(0 \leqslant \theta \leqslant n)$ if there are exactly θ planes in $\mathcal{D}(P)$ that lie strictly below q. The θth level of $\mathcal{A}(\mathcal{D}(P))$, denoted by λ_θ, is the closure of the points on the planes of $\mathcal{D}(P)$ whose levels are exactly θ in $\mathcal{A}(\mathcal{D}(P))$. Thus, λ_θ is a collection of faces which form a polyhedral terrain. In other words, a line perpendicular on the xy-plane intersects λ_θ at exactly one point. Further information on geometric duality and levels of an arrangement of the duals of a set of points in $\mathbb{R}^3$ are available in [3].

Circular range counting query

We now describe the preprocessing of the points in $P \in \mathbb{R}^2$ such that given any arbitrary circular range, the number of points inside it can be reported quickly.

We project the points in P on an unit paraboloid Π in $\mathbb{R}^3$ with origin of the coordinate system as vertex of the paraboloid. Thus, for a point $p = (x, y) \in P$, the transformed point is $p' = (x, y, x^2 + y^2)$. We name this set of transformed points as P'. Next, we consider the arrangement $\mathcal{A}(\mathcal{D}(P'))$ of the duals of points in P', and construct the levels of arrangement as follows:

Use an array of size n whose θth element contains the pointer to a data structure storing the faces of the terrain representing the level λ_θ.

For any query circle χ (in $\mathbb{R}^2$), a plane (say χ') can pass through the projection of χ on the paraboloid Π. Thus, the subset of points lying inside the circle χ (in $\mathbb{R}^2$) is same as those of the projected points below the plane χ' (in $\mathbb{R}^3$). This can be obtained in $O(\log n)$ time using the preprocessed data structure of size $O(n^3)$, and this preprocessing needs $O(n^3)$ time [1,2]. It may be mentioned that, the preprocessing time and space of the circular range query algorithm can be reduced somewhat, approximately by a factor $\log^3 n$, keeping the query time unchanged [11].

4. Semicircular range counting query

4.1. Preprocessing

We sort the points in P with respect to their x- and y-coordinates separately, and create two hight-balanced binary trees, namely T_1 and T_2. We describe the construction of T_1. The same procedure is adopted for the construction of T_2.

The root of T_1 corresponds to the entire set P. The points attached to the nodes at the next level are obtained by splitting the points in P into two subsets of almost equal size with a vertical line ℓ through the median x-coordinate of the point-set P. The splitting continues and the nodes of the tree are defined recursively in this manner until the size of a set becomes unity. The leaf nodes contain the respective points in P. Each non-leaf node v contains the following information:

(i) the set of points P_v attached to that node,
(ii) the vertical line ℓ_v splitting the points in P_v,
(iii) an integer field $\rho_v = |P_v|$, and
(iv) pointer to a secondary structure S_v storing the arrangement $\mathcal{A}(\mathcal{D}(P'_v))$, where P'_v is the set of transformed points in $\mathbb{R}^3$ corresponding to the points in P_v.

This is needed for the circular range query with the points in P_v. We need to further augment the secondary structure using the following lemma.

Lemma 1. *Let v be a node in T_1, and w be a successor of node v. A cell in S_v is completely contained in exactly one cell of S_w.*

With each cell $C \in S_v$, we attach two pointers, $cell\_ptr_L$ and $cell\_ptr_R$. These point to the cells $C_L \in S_u$ and $C_R \in S_w$, respectively, in which the cell C is contained. Basically, this can be done by attaching the pointers with each face $f \in \mathcal{D}(P'_v)$. If the face f is a part of a face $f^* \in C_L$, then its $cell\_ptr_L$ points to f^*. We draw a vertical line at a bounding vertex of f in downward direction. Let it hits the face $f^{**} \in C_R$. The $cell\_ptr_R$ points to f^{**}. If f is a part of a face in C_R, the $cell\_ptr_R$ and $cell\_ptr_L$ are set in a similar manner.

Lemma 2. *The time and space required for creating and storing the preprocessed data structure T are both $O(n^3)$.*

Proof. The initial binary tree T_1 can be constructed in $O(n \log n)$ time and space. The number of nodes at

ith level of T_1 is 2^i, and each of them contains $n/2^i$ points, $i = 0, 1, \ldots, \log n - 1$. For each non-leaf node v at level i, the size of the secondary structure S_v is $O((n/2^i)^3)$, and it can be constructed from the point set assigned to that node in $O((n/2^i)^3)$ time. So, the total time and space required for creating the secondary structures for all nodes in T is $O(\sum_{i=0}^{\log n-1} 2^i (n/2^i)^3) = O(n^3)$.

In the last stage of preprocessing, we use topological line sweep to set $cell\_ptr_L$ and $cell\_ptr_R$ attached to each edge of S_v. This requires an additional $O(n^3)$ amount of time. □

In a similar fashion, using the points in P sorted by their y-coordinates, we construct the binary tree T_2 in which the splitting lines are horizontal. We also attach secondary data structures to all non-leaf nodes of T_2 as described for T_1.

4.2. Query

Let χ be a semicircle specified by a tuple (ℓ, ϕ), where $\ell = [a, b]$ is a line-segment describing the diameter of the semicircle, and $\phi = 0$ or 1 depending on whether the required semicircle is to the same/different side of the origin with respect to the line containing ℓ. Our objective is to count number of points of P inside χ.

Let ℓ is not parallel to any of the two axes. We draw a horizontal and a vertical chord of the semicircle from a and b, respectively, as shown in Fig. 2. This decomposes χ into two arcs. We use the term *arc range* to indicate the region enclosed by an arc and the corresponding chord. Thus, we need to solve the counting query for two *arc ranges* and one *triangular range*. We shall refer an arc range having horizontal (resp., vertical) supporting chord as the *h-arc range* (resp., *v-arc range*). Observe that if ℓ is horizontal (resp., vertical) then the we get only an *h-arc range* (resp., *v-arc range*), and no *triangular range*.

We now describe *v-arc range* counting query using T_1 data structure.

4.2.1. v-arc range counting query

Without loss of generality, we assume that, the query *v-arc range* corresponds to a circle χ, and is to the left of its supporting chord. The plane corresponding to the projection of χ on Π is χ', and its dual (point) is denoted as $\mathcal{D}(\chi')$.

We traverse the preprocessed data structure T_1 from its root with the query *v-arc range*. A global COUNT field (initialized with 0) is maintained during the traversal. When the search is finished, this field will contain number of points lying inside the query *v-arc range*.

We first locate the cell containing $\mathcal{D}(\chi')$ in the secondary structure attached to the root node of T_1. During subsequent traversal, when search moves from a node u to one of its children, the cell containing $\mathcal{D}(\chi')$ in the secondary structure attached to that child of u are reached using $cell\_ptr_L$ or $cell\_ptr_R$ in O(1) time. When the search reaches a node u and u is leaf node, the COUNT field is incremented by one if the corresponding point lies inside χ. If u is a non-leaf node, then we need to consider the following three cases:

Case 1. The chord supporting the *v-arc range* matches with the split line attached to node u: We reach the cell containing $\mathcal{D}(\chi')$ in the secondary structure attached to the left child of u to determine the number of points lying within χ among the points attached to that node, and the COUNT field is increased by that number. We need not proceed further and the search stops.

Case 2. The supporting chord lies to the right of the split line attached to node u. Here, two subcases may be considered depending on whether the split line attached to node u cuts the *v-arc range*, or not.

In the first subcase, the circular range counting query is performed with the circle χ among the points which are attached to the left child of u, and the result is added to COUNT; next the search proceeds to the right child of u in T_1. In the second subcase, the search proceeds

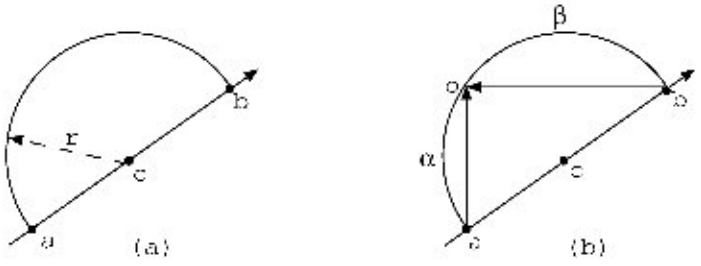

Fig. 2. Semicircular range and its decomposition.

to the right child of u without performing any circular range query.

Case 3.The supporting chord lies to the left of the split line. The processing of this case is exactly the same as in Case 2.

Lemma 3. *v-arc range counting query can be solved in* $O(\log n)$ *time.*

Proof. At the root node we spend $O(\log n)$ time for the point location query to identify the cell of the secondary structure containing the point $\mathcal{D}(\chi')$. At each of the subsequent move in the search path, we spent $O(1)$ time for this purpose. The number of points inside χ among the set of points attached to that node is obtained by observing the level of the cell containing $\mathcal{D}(\chi')$. This needs $O(1)$ time. As the height of T_1 is $O(\log n)$, worst case counting query time for a *v-arc range* is $O(\log n)$. □

The same method applies for the *h-arc range* counting query in the data structure T_2. Thus, we have the following theorem.

Theorem 1. *Let P be a set of n points in the plane. We can preprocess P in $O(n^3)$ time and space such that an arbitrary semicircular range counting query can be solved in $O(\log n)$ time.*

5. Chromatic distribution query

In this section, we shall concentrate on our main problem, where each member of P is assigned one of the m given colors, and the aim is to answer the queries Q1 and Q2 for an arbitrary query straightline segment, and a given value k.

Let $P_i \in P$ be the set of points in P having color i, $i = 1, 2, \ldots, m$. Apart from the construction of T_1 and T_2 with the set of points P, we also construct T_1^i and T_2^i with the points P_i for each color i, as described in the earlier section. The necessary data structures for rectangular range counting query is also to be prepared for each set P_i. But, the total time and space complexities for the preprocessing is dominated by those for P. Thus, we have the following lemma:

Lemma 4. *The preprocessing time and space complexities for the chromatic distribution query are both* $O(n^3)$.

Let $\sigma = [a, b]$ be the query line segment. This creates a corridor around σ. We have to find a width w such that the region $R(w)$ (the hippodrome of width w), as defined in Section 1, contains exactly k points of P.

After computing w, we need to find the number of points of each color inside the region $R(w)$. This answers both the versions of chromatic distribution query. As mentioned earlier, we would use P and P' to denote the given set of points and their projections on Π, respectively.

5.1. Computing the width w

Consider the levels of arrangement $\mathcal{A}(\mathcal{D}(P))$ (in $\mathbb{R}^2$). Let ℓ_σ denote the line containing the segment σ. Locate the point $\mathcal{D}(\ell_\sigma)$ in $\mathcal{A}(\mathcal{D}(P))$, and draw a vertical ray from the point $\mathcal{D}(\ell_\sigma)$ downwards to consider the subset of edges of $\mathcal{A}(\mathcal{D}(P))$ that are hit by the ray. These edges are at different levels of $\mathcal{A}(\mathcal{D}(P))$. Let us refer this set by E. Let $e \in E$ be an edge at level θ, and $\mathcal{D}(p)$ $(p \in P)$ be the line containing the edge e. We draw a line parallel to σ at the point p in the primal plane. Let w be the distance of this line from $\ell(\sigma)$. As mentioned earlier, we can compute the number of points in the hippodrome $R(w)$ around σ by performing one rectangular range counting query around σ, and two semicircular range counting queries at the two end points of σ. In [6], a binary search is applied among the members in E to compute the height of a rectangle on σ that contains exactly k points. We apply the same method to identify an $e_i \in E$ such that the corresponding point in the primal plane decides an width w of the hippodrome which contains either exactly k points or a pair of edges $e_i, e_{i+1} \in E$ such that the corresponding counts are less than k and greater than k, respectively. In the former case, w is the width of the hippodrome, and the search stops. In the latter case, we need to perform the same exercise with the edges of $\mathcal{A}(\mathcal{D}(P))$ intersected by the vertically upward ray from the point $\mathcal{D}(\ell_\sigma)$. If this also fails to settle the width of the hippodrome such that it contains exactly k points of P, we may need to adjust the diameter of the semicircles at both the ends of σ as follows. Note that, during this adjustment of width, the number of points in the rectangle around σ will not change.

Suppose we have obtained two values (say w_1 and w_2) of the width of hippodrome such that the count corresponding to w_1 (resp., w_2) is less (resp., greater) than k. Now, consider the set of semicircles with center at the left end of σ and radius equal to all real values from w_1 to w_2. Now we have the following observations:

(1) If we choose any width $w \in [w_1, w_2]$, the number of points in the rectangle of width $2 \times w$ around σ will be same.

(2) The dual of the planes in $\mathbb{R}^3$ corresponding to the circles with center at a and radii in $[w_1, w_2]$ lie on a line perpendicular to the xy-plane (by Result 1).

Consider the circle χ corresponding to w_1 at one end of σ. The COUNT field indicates the number of points inside the hippodrome $R(w_1)$, which is surely less than k. Let χ' be the image of χ on the unit paraboloid Π, and the point $\mathcal{D}(\chi')$ be the dual of the plane corresponding to it. We identify $\mathcal{D}(\chi')$ in the secondary structure attached to the root node of T_1 and T_2. Next, we identify the nodes of T_1 (resp., T_2) in which we need to perform the *v-arc* (resp., *h-arc*) range query. In order to obtain the radius $w^* \in [w, w']$ such that the COUNT attains the exact value k, we need to perform binary search on the planes in $\mathcal{A}(\mathcal{D}(P'))$ attached with root node of T_1 (resp., T_2) which are intersected by the upward vertical ray shot from point $\mathcal{D}(\chi')$ in the dual plane. As mentioned in Section 4.2.1, this can easily be performed in $O(\log n)$ time. If this also can not settle the desired width of the hippodrome, then we may have to perform the same exercise at the other end-point b of the line-segment σ.

Lemma 5. *The time required to find the desired width w such that the region $R(w)$ contains exactly k points is* $O(\log^2 n)$.

Proof. At each step of the binary search, (i) we choose an edge from the set E and construct the corresponding hippodrome around σ, and then (ii) we find the number of points inside that region by applying two rectangular range counting queries and two semicircular range counting queries. Since each of these queries takes $O(\log n)$ time, the lemma follows. □

After determining the width w so that the corresponding region $R(w)$ contains k nearest neighbors of σ, we find number of points of color i $(1 \leqslant i \leqslant m)$ in the same region $R(w)$ using the same technique on the respective data structure for point set P_i. As the number of colors m is assumed to be constant, this takes $O(\log n)$ time. Thus, we have the final result:

Theorem 2. *Given a set of n colored points in $\mathbb{R}^2$ with a fixed m $(1 \leqslant m \leqslant n)$ number of colors, it can be preprocessed in $O(n^3)$ time and space such that for an arbitrary query line segment σ chromatic distribution of k-nearest neighbor of σ can be determined in $O(\log^2 n)$ time, where k is an input at the query time.*

References

[1] P.K. Agarwal, J. Erickson, Geometric range searching and its relatives, in: B. Chazelle, J.E. Goodman, R. Pollack (Eds.), Advances in Discrete and Computational Geometry, Proc. of the 1996 AMS–IMS–SIAM Joint Summer Research Conference on Discrete and Computational Geometry: Ten Years Latter, in: Contemporary Mathematics, vol. 223, Amer. Math. Soc., 1996, pp. 1–56.

[2] B. Chazelle, M. Sharir, E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, Algorithmica 8 (1992) 407–429.

[3] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer, Berlin, 1987.

[4] K. Fukunaga, P.M. Narendra, A branch-and-bound algorithm for computing k-nearest neighbors, IEEE Transaction on Computers 24 (1975) 750–753.

[5] T. Graf, K. Hinrichs, Algorithms for proximity problems on colored point sets, in: Proc. 5th Canadian Conference on Computational Geometry, 1993, pp. 420–425.

[6] P.P. Goswami, S. Das, S.C. Nandy, Triangular range counting query in 2D and its application in finding k nearest neighbors of a line segment, Computational Geometry: Theory and Applications 29 (2004) 163–175.

[7] P. Gupta, R. Janardan and M. Smid, Efficient algorithms for generalized intersection searching on non-iso-oriented objects, in: Proc. 10th Annual ACM Symposium on Computational Geometry, 1994, pp. 369–378.

[8] Q. Jiang, W. Zhang, An improved method for finding nearest neighbors, Pattern Recognition Letters 14 (1993) 531–535.

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

[10] D.M. Mount, N. Netanyahu, R. Silverman, A.Y. Wu, Chromatic nearest neighbor searching: A query sensitive approach, Computational Geometry: Theory and Applications 17 (2000) 97–119.

[11] J. Matoušek, Range searching with efficient hierarchical cutting, in: Proc. 8th Annual Symp. on Computational Geometry, 1992, pp. 276–285.