Computing Closest Points for Segments

Sergei Bespamyatnikh*

Abstract

We consider the proximity problem of computing for each of n line segments the closest point from a given set of n points in the plane. We show that it can be solved in (i) $O(n^{4/3}2^{O(\log^* n)})$ time, and (ii) $O(n \log^2 n)$ time for the case of disjoint segments.

1 Introduction

We consider the following problem that we call *closest points problem.* Let P be a set of n points and S be a set of n segments in the plane. For each segment in S, compute its closest point in P.

The closest points problem is a generalization of *Hopcroft's problem* [4, 6, 8]: determine whether any point from a given set of n points lies on any of n given lines. Matoušek [8] gave an algorithm for Hopcroft's problem with $O(n^{4/3}2^{O(\log^* n)})$ running time. We present an algorithm for the closest points problem with the same running time, improving the result of Agarwal and Procopiu [3] by a factor $\log^k n, k > 1$. The running time is close to optimal in a class of algorithms based on partition trees [6]. For Hopcroft's problem Erickson [6] has shown that any algorithm based on partition trees, must take $\Omega(n^{4/3})$ time.

The closest points problem also generalizes the problem of finding *nearest foreign neighbors* [11, 1] where points are given instead of segments. The nearest foreign neighbors problem can be solved in optimal $O(n \log n)$ time. The on-line version of the problem, where the points whose query for a closest neighbor are unknown a priori, is the classical *post office* problem [7]. An optimal solution uses point location in the Voronoi diagram [10].

Bespamyatnikh and Snoeyink [3] considered the special case of the problem where segments are disjoint. and explored the on-line version of the closest points problem for a query segment. The main idea in [3] to find closest points for disjoint segments is the reduction to the special case of the on-line problem in which query segments are outside the convex hull of the points. They devised an algorithm that builds in $O(n \log n)$ time a data structure of size $O(n)$ so that the nearest point to a query segment can be found in $O(\log n)$ time, which is optimal. Based on this algorithm, they gave an $O(n \log^3 n)$ algorithm for the closest points problem.

In this paper we propose an extension of this approach for the closest points problem that combines a sweep technique with a dynamic maintenance of points for segment queries. The algorithm for the closest points problem runs in $O(n \log^2 n)$ time improving previous result [3] by a factor $\log n$. The closest points algorithm can be used to solve the following problems in the same time.

- Let R be a set of n disjoint red segments and B be a set of n disjoint blue segments in the plane. 1) For each red segment compute the closest blue segment and vice versa. 2) Compute the Hausdorff distance between R and B which is the maximum of the smallest distance from a red point to a blue point and the smallest distance from a blue point to a red point.
- Given n disjoint, colored line segments in the plane. For every segment, compute the closest segment of a different color.

2 General Segments

We show that the closest points for segments can be found in time matching the best bound for Hopcroft's problem [8].

Theorem 1 *Let P be a set of n points and S be a set of n segments in the plane. For all segments in S their closest points in P can be found*

*Department of Computer Science, Duke University, Box 90129, Durham, NC 27708, USA. E-mail: besp@cs.duke.edu

in $O(n^{4/3}2^{O(\log^* n)})$ time.

We use Matoušek's approach for solving Hopcroft's problem. The main difference is that the duality cannot be used directly since the dual problem is not the same. Let (n,m)-problem be the problem of computing closest points in a given set of n points for each of m given segments in the plane.

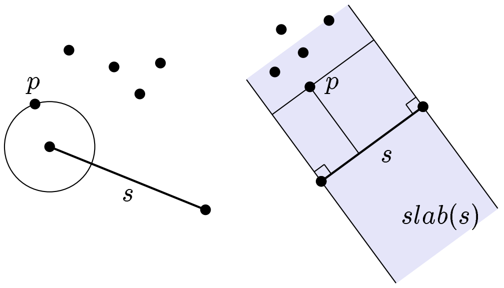
Lemma 2 *An (n,m)-problem can be solved in time $T(n,m)$ that can be bound as follows. Let r be an integer parameter.*

(i) *If $n \geq m$ then $T(n,m) = O((n+mr)\log n + r^2 T(n/r^2, m/r))$ for any $r \in [1, \min(\sqrt{n}, m)]$.*

(ii) *If $n < m$ then $T(n,m) = O((nr + m\log r)\log n + r^2 T(n/r, m/r^2))$ for any $r \in [1, \min(n, \sqrt{m})]$.*

3 Disjoint Segments

We use $d()$ to denote the distance between two points a and b, $d(a,b)$, and the distance between sets of points $d(A,B) = \inf_{a\in A, b\in B}\{d(a,b)\}$. Let $c(s,P)$ denote the closest point of a segment s to a set of points P, i.e $d(s, c(s,P)) = d(s,P)$. For a segment s and a subset $P' \subseteq P$, a point $p \in P'$ is a *candidate* for the closest point of s in P' if either $p = c(s,P)$ or $c(s,P) \in P \setminus P'$. For a segment s, the *slab of s*, or *slab*(s), is the region bounded by the perpendiculars to s through the endpoints of s, as shaded in Fig. 1. Wlog we assume that there are no vertical segments in S.



a) b)

Fig. 1: p is the closest point to s.

Observation 3 *The distance d(s,P) between a segment s and a point set P is achieved by either 1) an endpoint of s and its closest neighbor in P or by 2) an interior point of s and a point p in the slab of s. In the second case p is hit when the segment s is translated in the slab.*

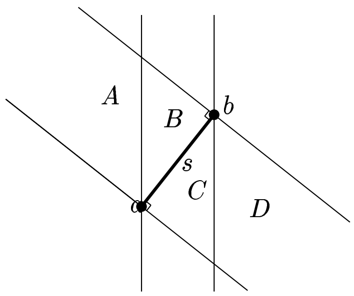


Fig. 2: Four regions.

Let $x_{\min}$ and $x_{\max}$ be the smallest and the largest x-coordinates of a point in s. Let $A(s)$ be the set of points of *slab*(s) whose x-coordinates are less than or equal to $x_{\min}$. Let $B(s)$ be the set of points of *slab*(s) that lie above or on s. Similarly, we define the sets $C(s)$ and $D(s)$, see Fig. 2. We show that the closest points problem can be reduced to the following dynamic problems with segment queries.

Problem A. Let P be a set of n points and S be a set of n segments in the plane. For each segment s in S, compute the candidate for the closest point of s in the set $A(s) \cap P$.

Problem B. Let P be a set of n points and S be a set of n segments in the plane. For each segment s in S, compute its closest point in the set $B(s) \cap P$.

Lemma 4 *The closet points for disjoint segments can found in $O(n\log n + T_A(n) + T_B(n))$ time where $T_A(n)$ and $T_B(n)$ are the running times of solving the problems A and B, respectively.*

In what follows we solve the problems A and B using their dynamic versions.

3.1 Problem A

Theorem 5 *The problem A can be solved in time $T_A(n) = O(n\log^2 n)$.*

Proof:[Sketch] We sweep the plane by a vertical line. At any moment we store the sequence L of Voronoi cells of swept points intersecting the sweep

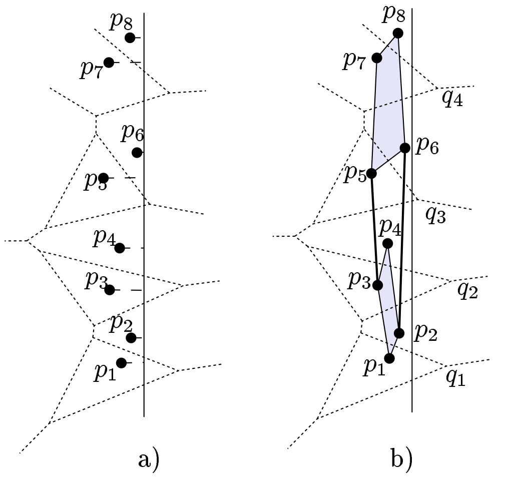


Fig. 3: a) The order of Voronoi cells. b) The bridges are marked by bold segments.

line. The order of the Voronoi cells intersecting l coincides with the order of y-coordinates of the corresponding points, see Fig. 3 (a). We maintain the binary search tree T for L equipped with bridges as in the dynamic convex hull structure of Overmars and van Leeuwen [9], see Fig. 3 (b). There are 3 types of events.

Point event. The sweep line passes over a point $p \in P$. The algorithm locates p in L, inserts p into L and T. Some Voronoi cells may no longer intersect the sweep line due to p. They are neighbors of p in L and can be found in $O(1)$ time per point. We delete them from L and T. Three consecutive points $a, b, c \in L$ can generate an event when l contains the point equidistant from a, b, c.

Segment endpoint event. The sweep line passes over the left endpoint a of a segment $s = [a, b] \in S$. The segment query can be answered applying search on the tree T with discarding operations as in [3].

Intersection point event. The sweep line passes over the intersection point of two Voronoi edges formed by three consecutive points of L, for example the point q_3 formed by $p_4, p_5, p_6 \in L$ in Fig. 3 b). The algorithm deletes the middle point because its Voronoi cell has no points to the right of l. We process the deletion of the point from L and possible following deletions in the same way as in point event. ∎

3.2 Segment dragging

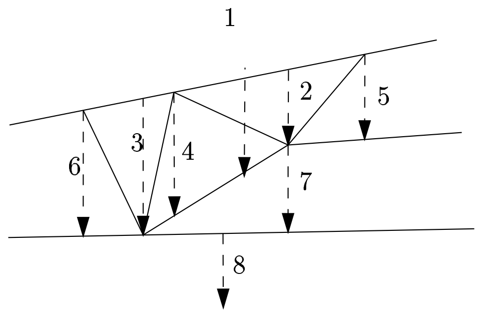


Fig. 4: Chain motion.

We consider the kinetic version of the problem where the points are swept by a chain falling down. The motion of the sweep chain is restricted so that at any moment of time only two links move downward (if the links are bounded segments then their common point goes downward; infinite links move by translation), see Fig. 4. A connected chain C of line segments is *x-monotone* if any vertical line intersects C in at most one point. We assume that the x-monotone chain extends to infinity on both ends.

Kinetic Segment Dragging. Let C be an x-monotone polygonal chain moving continuously downward so that between two events at most one vertex and two edges (maybe infinite) of C move. C has $O(n)$ segments at any time (the number of segments in C varies during the motion), see Fig. 4. Let P be a set of n points in the plane. Maintain a data structure for the subset of points in P above C so that, for a query segment $s \subset C$, its closest point in $P \cap B$ can be found efficiently.

There are two types of events.

Point Event. The chain C passes over a point of P. The algorithm inserts the point into a data structure.

Segment Event. The chain C contains a segment of S. The algorithm answers the query for the segment.

First we generate the chain and its motion by extending segments. A set S' of segments/rays is an *extension* of a set S of disjoint segments in the plane if

1) each segment/ray of S' contains a segment of S, and

2) the segment/rays of S' have disjoint interiors, and
3) the set S' is maximal, i.e. the set S' is the only extension of S'.

Lemma 6 *An extension of a set of disjoint segments in the plane can be constructed in $O(n \log n)$ time.*

Lemma 7 *There is a data structure for kinetic segment dragging that can be maintained in $O(\log^2 n)$ time per event and answers a segment query in $O(\log^2 n)$ time.*

Proof: As the underlying data structure, we dynamically maintain the lower hull of the points above the chain C. Using the data structure of Overmars and van Leeuwen [9], this takes $O(\log^2 n)$ time per update, and it provides the lower hull in the form of a balanced binary tree T of the edges in sorted order according to x-coordinates: Each internal node corresponds to a subchain of the convex hull. The lower hull corresponding to a node v consists of some edges of lower hulls of the child nodes and the edge $bridge(v)$ between them. We store the following information associated with a node v:
• the bridge $bridge(v) = [b_1(v), b_2(v)]$,
• $x_{\min}(v)$ and $x_{\max}(v)$, the smallest and largest x-coordinate of points below v.
• $x_{\text{cut}}(v)$, a real number separating x-coordinates of the child nodes.

The items $x_{\min}, x_{\max}$ and x_{cut} can be maintained in $O(\log n)$ time per update preserving the update time $O(\log^2 n)$.

It remains to show how to answer the segment query. Let $s = [a, b]$ be the query segment. The chain C contains s. There is a canonical representation of the points of P above s. It includes $O(\log n)$ nodes in T and they can be found in $O(\log n)$ time. Let V be the set of these nodes. Let $\alpha(s)$ be the slope of the segment s. Without loss of generality the slope $\alpha(s) \in [0, \pi/2]$, see Fig. 5 (a). Let $\beta(v) \in (-\pi/2, \pi/2)$ be the slope of the bridge vector $b_1(v)b_2(v)$ corresponding to a node v. We apply breadth first search to find the closest point to s in the region of $P \cap B$.

To carry out the search, the algorithm pops up all nodes of V, replaces them by their children, prunes the set of the children U and pushes result to V. Let $lson(v)$ and $rson(v)$ be the left and right sons of v. We break up the nodes of V into 3 groups, see Fig. 5.

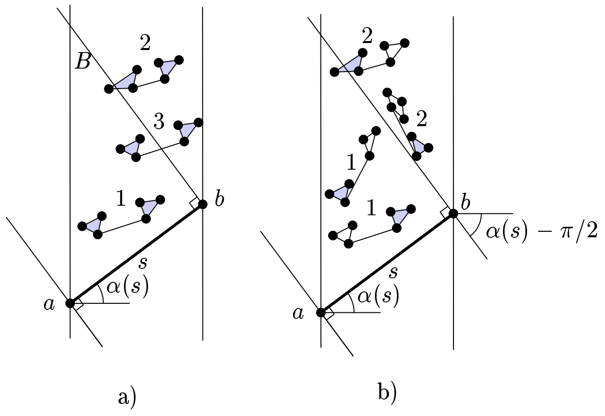


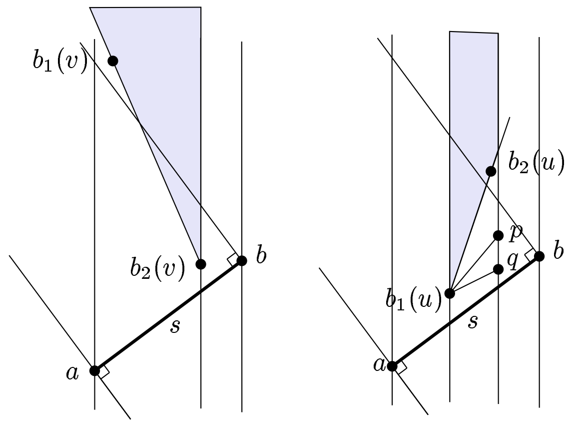
Fig. 5: a) 5 groups of V. b) Prune step. Removed nodes correspond to the light polygons.

1. $bridge(v) \subset slab(s)$. Both points $b_1(v)$ and $b_2(v)$ belong to the slab of s.
2. $bridge(v) \cap slab(s) = \emptyset$. The bridge of v is outside the slab of s.
3. Only one of the bridge endpoints is in the slab of s. In this case the right side of $slab$(s) separates the bridge of v.

We describe the prune step, illustrated in Fig. 5 (b) and Fig. 6.

1. For a node v of the first group, remove one of the sons $lson(v), rson(v)$ whose bridge endpoint is farther from the segment s (it can be determined by comparing the slopes $\alpha(s)$ and $\beta(v)$).
2. For a node v of the second group, remove one of the sons $lson(v), rson(v)$ depending on slope $\beta(v)$. If $\beta(v) \leq \alpha(s) - \pi/2$ then remove $lson(v)$. Otherwise remove $rson(v)$.
3. For a node v of the third group whose right bridge endpoint is in the slab of s (i.e. $b_2(v) \in slab(s)$), remove $lson(v)$ since $b_2(v)$ is closer to s than any point below $lson(v)$, see Fig. 6 (a).
4. For two nodes u and v in the third group with two surviving sons, remove one of the sons using the following rule. Without loss of generality the node u is to the left of v. We show that the right son of u can be removed. Note that $b_1(u)$ and $b_1(v)$ are inside $slab$(s) by the previous discarding rule. We discriminate 2 cases. If the slope of the vector $b_1(u)b_1(v)$ (taken from $(-\pi/2, \pi/2)$) is at least $\alpha(s)$ then all the points below $rson(u)$ are

farther from s than $b_1(u)$, see $p = b_j(v)$ in Fig. 6 (a). Otherwise they are farther from s than $b_1(v)$), see $q = b_j(v)$ in Fig. 6 (b).



a) b)

Fig. 6: a) Prune rule 3: points below $lson(v)$ are in shaded wedge. b) Prune rule 4: points below $rson(u)$ are in shaded polygon.

We analyze the running time of the search algorithm. There are $O(\log n)$ nodes in V at the beginning. Each node of V except at most one node of the third group brings at most one son to U. Since the height of T is $O(\log n)$, the complexity of V is always $O(\log n)$ and the total number of visited nodes is $O(\log^2 n)$. The search time is $O(\log^2 n)$ because each node is processed in a constant time. ∎

We conclude with the main result.

Theorem 8 *Given n points and n disjoint segments in the plane, the closest point to every segment can be computed in $O(n \log^2 n)$ time.*

4 Extensions

The closest points algorithm can be used to solve other applications [3]. The common feature of their definitions is *nearest foreign neighbors* NFN. The points in the closest points problem can be extended to segments. It leads to two groups of segments and we color them.

Two colored asymmetric NFN. Given n disjoint red segments and n disjoint blue segments in the plane, for each blue segment compute its nearest red segment.

For sets B and R of blue and red segments, let $di(B, R)$ be the smallest distance from a blue segment to its nearest red segment.

Two colored symmetric NFN. Given n disjoint red segments and n disjoint blue segments in the plane, for each segment compute its nearest segment of a different color.

Hausdorff distance. Compute the Hausdorff distance between n disjoint red segments and n disjoint blue segments in the plane. The Hausdorff distance is the maximum of the distances $di(B, R)$ and $di(R, B)$.

Colored NFN. Given n disjoint, colored line segments in the plane, for each segment compute its nearest segment of a different color.

Theorem 9 *All of the above problems can be solved in $O(n \log^2 n)$ time.*

References

[1] A. Aggarwal, H. Edelsbrunner, P. Raghavan, and P. Tiwari. Optimal time bounds for some proximity problems in the plane. *Inform. Process. Lett.*, 42(1):55–60, 1992.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, 1979.

[3] S. Bespamyatnikh and J. Snoeyink. Queries with segments in Voronoi diagrams. *Comput. Geom. Theory Appl.*, 16(1):23–33, 2000.

[4] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[6] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.*, 16:389–418, 1996.

[7] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, Reading, MA, 1973.

[8] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.

[9] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[10] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 151–162, 1975.

[11] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.